Financial Analytics (MGT3012)

Project Report

MORTAGAGE LOAN AUTOMATION

By

19MIA1063	AISHWARYA S
19MIA1073	KEERTHANA MADHAVAN
19MIA1084	PODALAKURU SAHITHYA

M.Tech Integrated Computer Science and Engineering
With specialization in Business Analytics

Submitted to

Dr. Jyotirmayee

VITBS, VIT Chennai

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

April 2023

DECLARATION

We hereby declare that the report titled "**Mortgage Loan Automation"** submitted by us to VIT Chennai is a record of bona-fide work undertaken by us under the supervision of **Dr**.**Jyotirmayee**, VIT Business School, Vellore Institute of Technology, Chennai.

Signature of Candidates

AISHWARYA S 19MIA1063

KEERTHANA MADHAVAN 19MIA1073

PODALAKURU SAHITHYA 19MIA1084

CERTIFICATE

Certified that this project report entitled "**Mortgage Loan Automation"** is a bonafide work of **AISHWARYA S 19MIA1063, KEERTHANA MADHAVAN 19MIA1073** and **PODALAKURU SAHITHYA 19MIA1084** and they carried out the Project work under my supervision and guidance for **Financial Analytics(MGT3012)**

Dr. Jyotirmayee

VITBS,VITChennai

ACKNOWLEDGEMENT

We would like to acknowledge that our assignment has been completed and we are ensuring that this was done by us and not copied.

In this accomplishment,we would like to express our special gratitude to Dr.Jyotirmayee, without her guidance and feedback it is not possible to complete this project.

Finally, we would like to thank our parents and friends who helped usa lot in finishing this assignment.

AISHWARYA S 19MIA1063
KEERTHANA MADHAVAN 19MIA1073
PODALAKURU SAHITHYA 19MIA1084

ABSTRACT

Loan prediction is a very common real-life problem that each retail bank faces at least once in its lifetime. If done correctly, it can save a lot of man hours at the end of a retail bank. Customers first apply for a home loan after that company validates the customer eligibility for the loan. The loan eligibility process (real time) can be automated based on customer details provided through for example filling an online application form. These details can be Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and many more. Hence the goal is to identify the customer segments that are eligible and in fact germane for loan amounts so that they can be specifically targeted. Ensuring the mortgage origination process goes smoothly is a considerable challenge. The ability to streamline loan processing, while ensuring data privacy, accuracy, and regulatory compliance is a game changer. Mortgage processing automation significantly enhances the lending process and enables financial institutions to do away with manual processing by routing forms and data automatically, reducing the complexity of processing applications, decreasing potential errors, and ensuring customers receive the best experience possible.

CONTENTS

1. Introduction

Mortgage Bank Loan Analytics with ARIMA and Machine Learning Mortgage Loans Analytics Banks can now use mortgage loan analytics using Data Science techniques. The system can provide detail information of the mortgage loans and the mortgage loan markets. It is a powerful tool for mortgage brokers to seek counterparties and generate trading interests and is useful for the CFOs to conduct what-ifs scenarios on the balance sheets.

2. Problem Statement

A glaring problem with the conventional mortgage loan origination process is the excessive use of paper-based approval procedures, complex spreadsheets, and outdated manual systems. Such systems are usually siloed and are marked by data management problems, creating more work for loan processing teams and opacity for the management and external examiners. This invariably elongates the decision-making cycles of lenders and is often the cause of borrower frustration.

These snags in the paper-based practices have paved the way for digitization of the ecosystem for gains such as a consistent lending cycle, auditability, and accuracy, among others. Today's automatic mortgage system allows streamlining of the entire loan data management system. It leverages Machine Learning capabilities to accurately classify documents, thus addressing the concern of disparate systems and facilitating reliable and consistent dataflows across all stages of the loan origination process.

3. Dataset

We have taken the dataset from kaggle

The dataset contains fields like: Loan ID, loan type, Loan program type, balance, Loan Size, Maturity, etc.

https://www.kaggle.com/datasets/gauravduttakiit/mortgage-bank-loan

4. Literature Survey

From this paper[1] , The objective of this research was to investigate if deployment of Machine Learning models can provide a better predictive capability than the conventional process used by lending institutions in Pakistan to make mortgage approval or rejection decisions so that the percentage of non-performing mortgage assets might be reduced. They used algorithms like Logistic Regression, Random Forest Classifier and Gradient Boosting Tree Classifier, and each model was tuned with its hyperparameters. Among all the algorithms they used , the gradient boosting tree classifier provided the best accuracy.

This paper [2] presents an explainable AI decision-support-system to automate the loan underwriting process by belief-rule-base (BRB). This system can accommodate human knowledge and can also learn from historical data by supervised learning. A business case study on automation of mortgage underwriting is demonstrated to show that the BRB system can provide a good trade-off between accuracy and explainability.

5. Code Implementation

We download the dataset and then import the necessary libraries first we started with preprocessing and visualization.

Creating loan analyst functions:

```python
class Loan:

    def __init__(self, rate, term, loan_amount, start=dt.date.today().isoformat()):
        self.rate = rate / 1200
        self.periods = term * 12
        self.loan_amount = loan_amount
        self.start = month_start(dt.date.fromisoformat(start) + dt.timedelta(31))
        self.pmt = npf.pmt(self.rate, self.periods, -self.loan_amount)
        self.pmt_str = f"${self.pmt:,.2f}"
        self.table = self.loan_table()

    def loan_table(self):
        periods = [self.start + relativedelta(months=x) for x in range(self.periods)]
        interest = [npf.ipmt(self.rate, month, self.periods, -self.loan_amount)
                    for month in range(1, self.periods + 1)]
        principal = [npf.ppmt(self.rate, month, self.periods, -self.loan_amount)
                     for month in range(1, self.periods + 1)]
        table = pd.DataFrame({'Payment': self.pmt,
                              'Interest': interest,
                              'Principal': principal}, index=pd.to_datetime(periods))
        table['Balance'] = self.loan_amount - table['Principal'].cumsum()
        return table.round(2)

    def plot_balances(self):
        amort = self.loan_table()
        plt.plot(amort.Balance, label='Balance')
        plt.plot(amort.Interest.cumsum(), label='Interest Paid')
        plt.grid(axis='y', alpha=.5)
        plt.legend(loc=8)
        plt.show()

    def summary(self):
        amort = self.table
        print("Summary")
        print("-" * 30)
        print(f'Payment: {self.pmt_str:>21}')
        print(f'{"Payoff Date:":19s} {amort.index.date[-1]}')
        print(f'Interest Paid: {amort.Interest.cumsum()[-1]:>15,.2f}')
        print("-" * 30)

    def pay_early(self, extra_amt):
        return f'{round(npf.nper(self.rate, self.pmt + extra_amt, -self.loan_amount) / 12, 2)}'

    def retire_debt(self, years_to_debt_free):
        extra_pmt = 1
        while npf.nper(self.rate, self.pmt + extra_pmt, -self.loan_amount) / 12 > years_to_debt_free:
            extra_pmt += 1
        return extra_pmt, self.pmt + extra_pmt
```

Menu

---

1. Start new loan
2. Show Payment
3. Show Amortization Table
4. Show Loan Summary
5. Plot Balances
6. Show size of payment to payoff in specific time
7. Show effect of adding amount to each payment
8. Quit

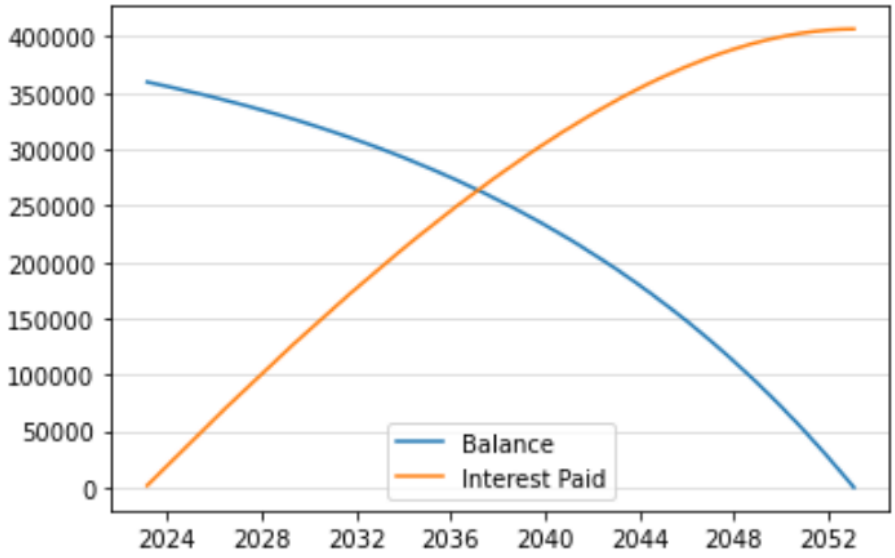
```
def display_menu():
    print("""
    Menu
    -----------------
    1. Start new loan
    2. Show Payment
    3. Show Amortization Table
    4. Show Loan Summary
    5. Plot Balances
    6. Show size of payment to payoff in specific time
    7. Show effect of adding amount to each payment
    8. Quit
    """)


def new_loan(rate, term, pv):
    loan = Loan(rate, term, pv)
    print("Loan initialized")
    sleep(.5)
    return loan


def pmt(loan):
    print(loan.pmt_str)


def amort(loan):
    print(loan.table)


def summary(loan):
    loan.summary()


def plot(loan):
    loan.plot_balances()


def pay_faster(loan):
    amt = float(input("Enter extra monthly payment: "))
    new_term = loan.pay_early(amt)
    print(f"Paid off in {new_term} years")


def pay_early(loan):
    years_to_pay = int(input("Enter years to debt free: "))
    result = loan.retire_debt(years_to_pay)
    print(f"Monthly extra: ${result[0]:,.2f} \tTotal Payment: ${result[1]:,.2f}")
```

```
action = {'1': new_loan, '2': pmt, '3': amort, '4': summar
          '5': plot, '6': pay_early, '7': pay_faster}


# program flow
def main():
    while True:
        display_menu()
        choice = input("Enter your selection: ")
        if choice == '1':
            rate = float(input("Enter interest rate: "))
            term = int(input("Enter loan term: "))
            pv = float(input("Enter amount borrowed: "))
            loan = Loan(rate, term, pv)
            print("Loan initialized")
            sleep(.75)

        elif choice in '234567':
            try:
                action[choice](loan)
                sleep(2)
            except NameError:
                print("No Loan setup")
                print("Set up a new loan")
                sleep(2)

        elif choice == '8':
            print("Goodbye")
            sys.exit()
        else:
            print("please enter a valid selection")


if __name__ == "__main__":
    main()
```

```
    Menu
    ----------------
    1. Start new loan
    2. Show Payment
    3. Show Amortization Table
    4. Show Loan Summary
    5. Plot Balances
    6. Show size of payment to payoff in specific time
    7. Show effect of adding amount to each payment
    8. Quit

Enter your selection: 3
No Loan setup
Set up a new loan
```

```
Enter your selection: 1
Enter interest rate: 5.875
Enter loan term: 30
Enter amount borrowed: 360000
Loan initialized

    Menu
    -----------------
    1. Start new loan
    2. Show Payment
    3. Show Amortization Table
    4. Show Loan Summary
    5. Plot Balances
    6. Show size of payment to payoff in specific time
    7. Show effect of adding amount to each payment
    8. Quit

Enter your selection: 2
$2,129.54

    Menu
    -----------------
    1. Start new loan
    2. Show Payment
    3. Show Amortization Table
    4. Show Loan Summary
    5. Plot Balances
    6. Show size of payment to payoff in specific time
    7. Show effect of adding amount to each payment
    8. Quit

Enter your selection: 3
            Payment            Interest  Principal    Balance
2023-03-01  2129.54              1762.5     367.04  359632.96
2023-04-01  2129.54  1760.7030531740681     368.83  359264.13
2023-05-01  2129.54  1758.8973087959678     370.64  358893.49
2023-06-01  2129.54  1757.0827237943495     372.45  358521.04
2023-07-01  2129.54  1755.2592548869939     374.28  358146.76
...             ...                 ...        ...        ...
2052-10-01  2129.54   51.37227837216121    2078.16    8414.90
2052-11-01  2129.54   41.19793540909152    2088.34    6326.56
2052-12-01  2129.54  30.973780558600364    2098.56    4228.00
2053-01-01  2129.54   20.69956994998599    2108.84    2119.16
2053-02-01  2129.54  10.375058518602795    2119.16      -0.00

[360 rows x 4 columns]
```

```
Enter your selection: 4
Summary
------------------------------
Payment:             $2,129.54
Payoff Date:        2053-02-01
Interest Paid:      406,632.94
------------------------------

    Menu
    -----------------
    1. Start new loan
    2. Show Payment
    3. Show Amortization Table
    4. Show Loan Summary
    5. Plot Balances
    6. Show size of payment to payoff in specific time
    7. Show effect of adding amount to each payment
    8. Quit

Enter your selection: 5
```




```
    Menu
    -----------------
    1. Start new loan
    2. Show Payment
    3. Show Amortization Table
    4. Show Loan Summary
    5. Plot Balances
    6. Show size of payment to payoff in specific time
    7. Show effect of adding amount to each payment
    8. Quit

Enter your selection: 6
Enter years to debt free: 10
Monthly extra: $1,845.00        Total Payment: $3,974.54
```

EXPLORATORY DATA ANALYSIS:

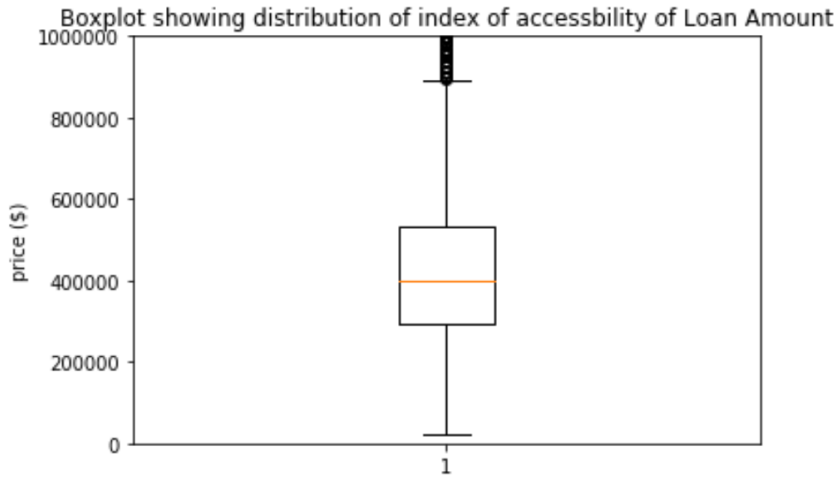
Print the 5th and 95th percentiles
Separating DataFrame subsets by State and selecting data per column

```
In [3]: # Print the 5th and 95th percentiles
print(data.quantile([0.05, 0.95]))
print(data['Loan Amount'].quantile([0.05, 0.95]))
print(data['Loan Amount'].quantile([0.1, 0.90]))

print(data['Loan Amount'].quantile([0.1, 0.90]))

St=['CT', 'FL', 'NJ', 'NY', 'PA']
print(St)

#DataFrame subsets by State
ct_data = data[data['State'].isin(['CT'])]
fl_data = data[data['State'].isin(['FL'])]
ny_data = data[data['State'].isin(['NY'])]
nj_data = data[data['State'].isin(['NJ'])]
pa_data = data[data['State'].isin(['PA'])]


#Selecting data per column
ct_lo_amount = ct_data.loc[:,('Created Date', 'First Name', 'Last Name', 'Loan Amount', 'City', 'unit_type_code', 'Loan Type')]
fl_lo_amount = fl_data.loc[:,('Created Date', 'First Name', 'Last Name', 'Loan Amount', 'City', 'Unit Type', 'Loan Type')]
nj_lo_amount = nj_data.loc[:,('Created Date', 'First Name', 'Last Name', 'Loan Amount', 'City', 'Unit Type', 'Loan Type')]
ny_lo_amount = ny_data.loc[:,('Created Date', 'First Name', 'Last Name', 'Loan Amount', 'City', 'Unit Type', 'Loan Type')]
pa_lo_amount = pa_data.loc[:,('Created Date', 'First Name', 'Last Name', 'Loan Amount', 'City', 'Unit Type', 'Loan Type')]
```

```
          Zip  Loan Amount  Estimated Value  Qualification FICO  \
0.05   7083.0     150000.0         205275.0               636.0
0.95  11756.0     840000.0        1400000.0               803.0

      Is Primary Wage Earner  Fix  ARM  CLTV  LoanInMonth
0.05                     0.0  0.0  0.0   0.0          3.0
0.95                     1.0  1.0  1.0  96.0         56.0
0.05    150000.0
0.95    840000.0
Name: Loan Amount, dtype: float64
0.1    200000.0
0.9    696500.0
Name: Loan Amount, dtype: float64
0.1    200000.0
0.9    696500.0
Name: Loan Amount, dtype: float64
['CT', 'FL', 'NJ', 'NY', 'PA']
```

Creating boxplot to see the distribution media and outliers
We can see that mid 50% loan amount between 340000 & 570000

```
In [4]: #creating boxplot to see the distribution median and outliers
plt.boxplot(data['Loan Amount'])
plt.title("Boxplot showing distribution of index of accessbility of Loan Amount")
plt.ylim(0.1,1000000)
plt.ylabel('price ($)')
plt.show()
```

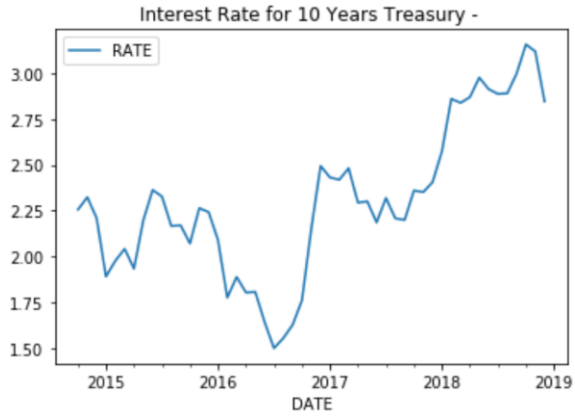


Load us 10 years treasury data

Check for missing values and find interest rate for 10 years treasury

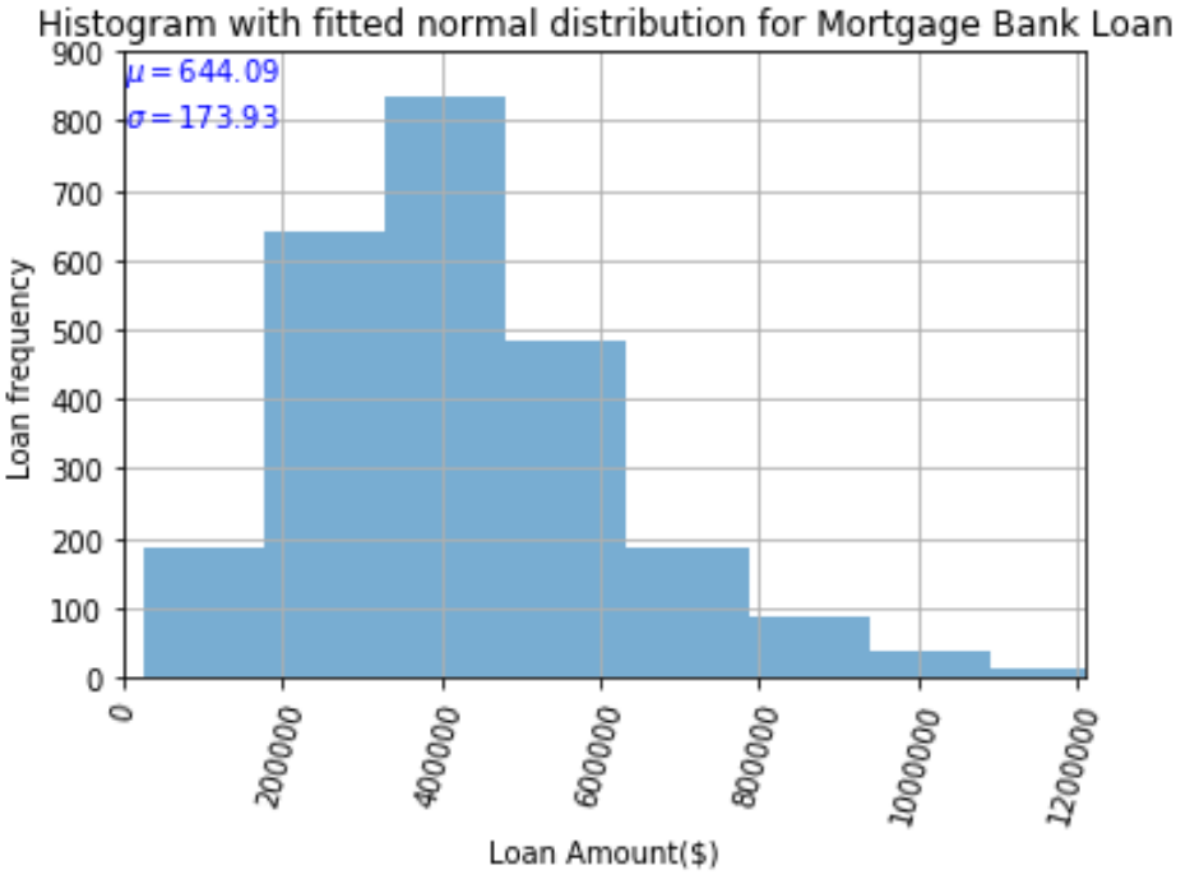
```
In [9]: US10Y.index = pd.to_datetime(US10Y.index)
        monthly_rate_data=US10Y.resample('M',how='mean').plot(title="Interest Rate for 10 Years Treasury - ")
```

```
C:\Users\achow\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: how in .resample()
the new syntax is .resample(...).mean()
```



Creating a histogram of 20 bins to show the distribution of parameters.

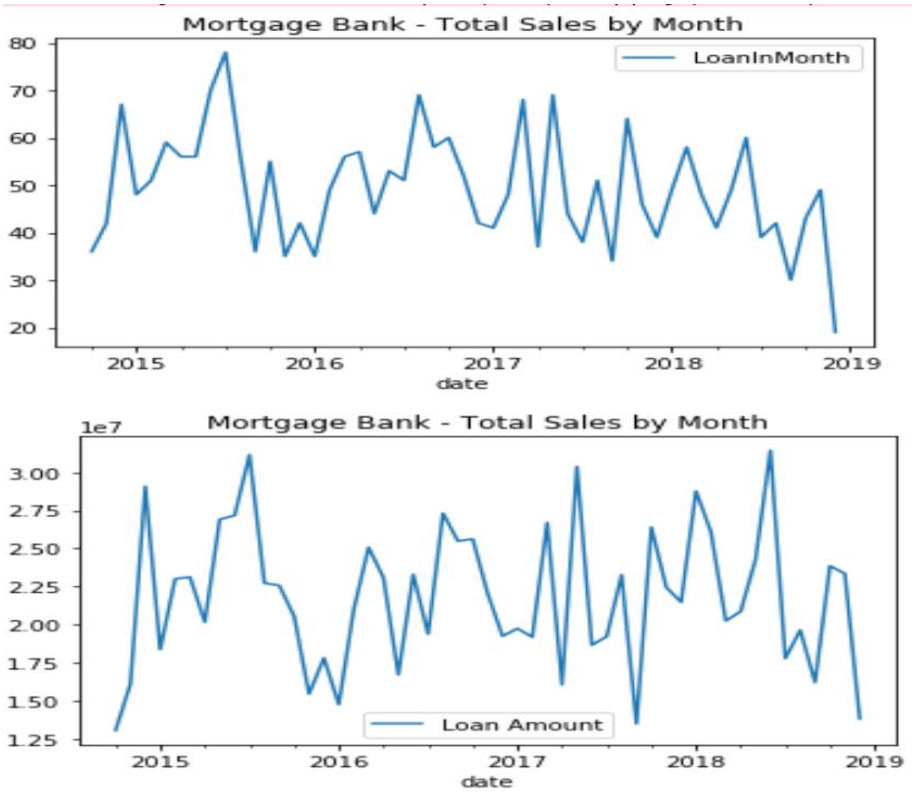
Plotting the histogram again with parameters mean and standard distribution.



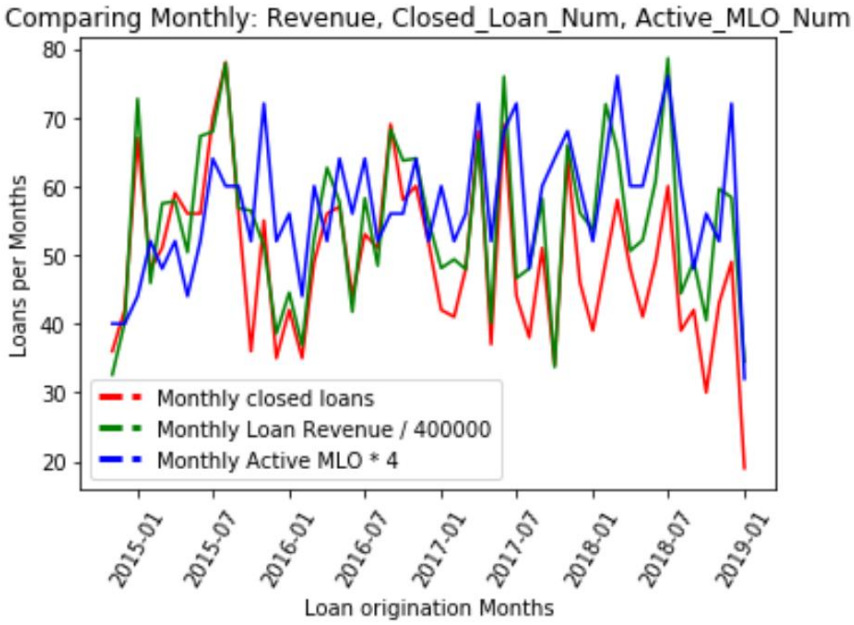
Examining kurtosis and skewness.

```
excess kurtosis of normal distribution (should be 0): 10.20423896299732
skewness of normal distribution (should be 0): 1.6231675369974472
mean :  644.086535155362
var  :  30252.443004451427
skew :  1.6231675369974472
kurt :  10.20423896299732
```

Now We Will Compare Monthly Revenue, Monthly Closed Loan Number and Active Mortgage Loan Originators. We will count number of Mlo actively Closing Loans on any given month



Mortgage Bank Monthly Sales Since October 2014. Sales varies between 12M & 33M per months. Summer seems to be high sales seasons for the Mortgage bank. Numbers of loans closed per months varies between 30 & 80.



As we know that number of producer is essential component in any given business. MLO

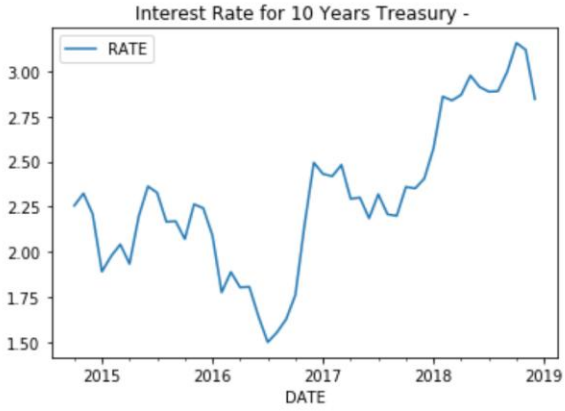
(Mortgage loan Originator) is core component in Mortgage business. Many MLO works indedendently and interect directly to clients, involve in marketting and grow their business. There could be many MLO in Mortgage Bank, but active MLO generate more reverue for the bank. As number of active MLO goes up, which will directly and positively impact numbers of loan closed per month, eventually mortgage revenue will go up. On the other hand, once number of active MLO goes down, mortgage revenue and number of loan per month goes down as well. By visualizing the graphs, we can see that monthly data of closed loan numbers , monthly revenue and active MLO numbers ber months, all moving at the same direction.

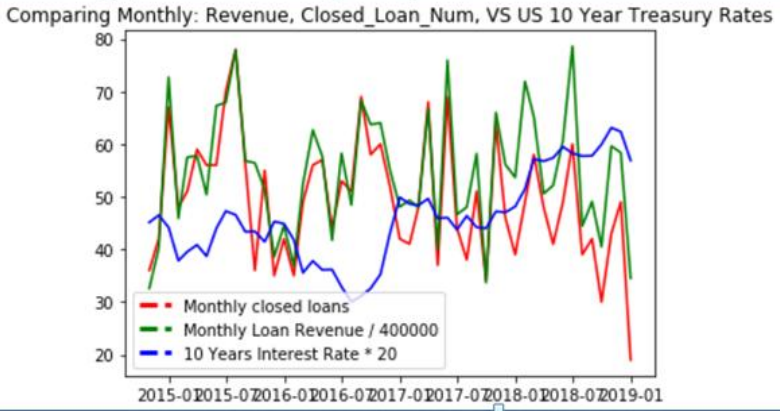
Let's find out interest rate effect on monthly closed loans and monthly revenue

```
In [14]: US10Y.index = pd.to_datetime(US10Y.index)
monthly_rate_data=US10Y.resample('M',how='mean').plot(title="Interest Rate for 10 Years Treasury - ")
print(monthly_rate_data)
```

```
C:\Users\achow\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: how in .resample()
the new syntax is .resample(...).mean()
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```





We can see the strong positive correlation between Monthly Closed Loans and Monthly Revenue. This graph also suggest that, as interest rates goes down, banks monthly revenue and numbers of loans increases, and when the Rates goes up, both Monthly Closed Loans and Monthly Revenue for the Mortgage bank decline. Pearson correlation coefficient between Monthly Interest & Monthly loans Closed Data is -0.334, which clearly proves that Monthly Interest Rates & Monthly loans Closed Data is negatively correlated.

In [16]:
```
def pearson_r(x, y):
    """Compute Pearson correlation coefficient between two arrays."""
    # Compute correlation matrix: corr_mat
    corr_mat = scipy.stats.pearsonr()

    # Return entry [0,1]
    return corr_mat[0, 1]
```

In [17]:
```
#scipy.stats.pearsonr(x, y)
%matplotlib inline

import numpy as np
import pandas as pd
import scipy
import scipy.stats as stats
import matplotlib.pyplot as plt
import sklearn

fico = data['Qualification FICO']
loan_amount=data['Loan Amount']
cltv_data = data['CLTV']
scipy.stats.pearsonr(cltv_data, fico)
r_loan_amount_fico = scipy.stats.pearsonr(loan_amount, fico)
r_cltv_data_fico = scipy.stats.pearsonr(cltv_data, fico)
r_loan_amount_cltv_data = scipy.stats.pearsonr(loan_amount, cltv_data)
r_monthly_loan_num_data_monthly_loan_rev = scipy.stats.pearsonr(monthly_loan_num_data, monthly_loan_rev_data)

# Print the result
print('Pearson correlation coefficient between FICO and Loan_Amount: ', r_loan_amount_fico)

print('Pearson correlation coefficient between FICO and CLTV: ', r_cltv_data_fico)

print('Pearson correlation coefficient between Loan Amount and CLTV: ', r_loan_amount_cltv_data)

print('Pearson correlation coefficient between Loan number and Loan Revenue: ', r_monthly_loan_num_data_monthly_loan_rev)
```

```
Pearson correlation coefficient between FICO and Loan_Amount:  (0.007679913675807394, 0.7010628450657441)
Pearson correlation coefficient between FICO and CLTV:  (-0.03706042959216819, 0.06386772845856088)
Pearson correlation coefficient between Loan Amount and CLTV:  (-0.006696250380445668, 0.7378394696435531)
Pearson correlation coefficient between Loan number and Loan Revenue:  (array([0.84472037]), array([6.63988048e-15]))
```

Acquire 1000 pairs bootstrap replicates of the Pearson correlation coefficient using the draw_bs_pairs() function you wrote in the previous exercise for CLTV data VS Qualification FICO Data and Monthly Loan_num_data VS. Monthly_loan_rev. Compute the 95% confidence interval for both using your bootstrap replicates. -We have created a NumPy array of percentiles to compute. These are the 2.5th, and 97.5th. By creating a list and convert the list to a NumPy array using np.array(). For example, np.array([2.5, 97.5]) would create an array consisting of the 2.5th and 97.5th percentiles.

In [18]:
```
pearson_r0=scipy.stats.pearsonr(cltv_data, fico)
pearson_r1=scipy.stats.pearsonr(loan_amount, fico)


# Print results
print('CLTV data VS Qualification FICO Data       :', pearson_r0)
print('Monthly Loan_Amount VS. FICO Data          :', pearson_r1)
```

```
CLTV data VS Qualification FICO Data       : (-0.03706042959216819, 0.06386772845856088)
Monthly Loan_Amount VS. FICO Data          : (0.007679913675807394, 0.7010628450657441)
```

DICKEY-FULLER TEST:

In [20]:
```
loan_rev_data=data[['Loan Amount']]
loan_rev_data['date'] = pd.DatetimeIndex(data['Created Date'])
loan_rev_data = loan_rev_data.set_index('date')
#monthly_loan_rev_data=loan_rev_data.resample('M', how=sum).plot(title="Mortgage Bank - Total Sales by Month",legend=True)


monthly_loan_rev_data= loan_rev_data.resample('M').sum()
print(monthly_loan_rev_data)
# Run the ADF test on the monthly_loan_rev_data series and print out the results
adfuller_loan_rev_data = adfuller(monthly_loan_rev_data['Loan Amount'], autolag='AIC')

print(adfuller_loan_rev_data)
# Just print out the p-value
print('The p-value of the test on loan_rev is: ' + str(adfuller_loan_rev_data[1]))
print('\n Print in different format')
print('Results of Dickey-Fuller Test:')
dfoutput = pd.Series(adfuller_loan_rev_data[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in adfuller_loan_rev_data[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
            Loan Amount
date
2014-10-31  13039283.00
2014-11-30  16097733.00
2014-12-31  29077334.00
2015-01-31  18361520.00
2015-02-28  22995652.00
2015-03-31  23113902.00
2015-04-30  20157614.00
2015-05-31  26908603.00
2015-06-30  27168128.00
2015-07-31  31152388.00
2018-10-31  23847863.16
2018-11-30  23346017.53
2018-12-31  13805150.00
(-4.49967715626648, 0.00019690419763896495, 10, 40, {'1%': -3.6055648906249997, '5%': -2.937069375, '10%': -2.606985625}, 1307.285137861741)
The p-value of the test on loan_rev is: 0.00019690419763896495

 Print in different format
Results of Dickey-Fuller Test:
Test Statistic                 -4.499677
p-value                         0.000197
#Lags Used                     10.000000
Number of Observations Used    40.000000
Critical Value (1%)            -3.605565
Critical Value (5%)            -2.937069
Critical Value (10%)           -2.606986
dtype: float64
```

Run the adf test on the monthly_rate_data series and print out the results
Print out the p-value

In [21]:
```
monthly_rate=US10Y.resample('M').mean()
monthly_rate_data=monthly_rate['RATE']
type(monthly_rate_data)

# Run the ADF test on the monthly_rate_data series and print out the results
adfuller_monthly_rate_data = adfuller(monthly_rate_data)
print(adfuller_monthly_rate_data)

# Just print out the p-value
print('The p-value of the test on monthly_rate_data is: ' + str(adfuller_monthly_rate_data[1]))


print('Results of Dickey-Fuller Test:')
dfoutput = pd.Series(adfuller_monthly_rate_data[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in adfuller_monthly_rate_data[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
(-1.396544767435691, 0.5839314748568241, 1, 49, {'1%': -3.5714715250448363, '5%': -2.922629480573571, '10%': -2.5993358475635153}, -34.97121939430423)
The p-value of the test on monthly_rate_data is: 0.5839314748568241
Results of Dickey-Fuller Test:
Test Statistic                 -1.396545
p-value                         0.583931
#Lags Used                      1.000000
Number of Observations Used    49.000000
Critical Value (1%)            -3.571472
Critical Value (5%)            -2.922629
Critical Value (10%)           -2.599336
dtype: float64
```

According to this test, p-value is very is higher than 0.05. We cannot reject the hypothesis that Monthly Interest Rate prices follow a random walk.

Random Walk

Are Interest Rates or Monthly Loan Returns Prices a Random Walk?

Most returns prices follow a random walk (perhaps with a drift). We will look at a time series of Monthly Sales Revenue, and run the 'Augmented Dickey-Fuller Test' from the statsmodels library to show that it does indeed follow a random walk. With the ADF test,

the "null hypothesis" (the hypothesis that we either reject or fail to reject) is that the series follows a random walk. Therefore, a low p-value (say less than 5%) means we can reject the null hypothesis that the series is a random walk. Print out just the p-value of the test (adfuller_loan_rev_data[0] is the test statistic, and adfuller_loan_rev_data[1] is the p-value). Print out the entire output, which includes the test statistic, the p-values, and the critical values for tests with 1%, 10%, and 5% levels.

Compute and print the autocorrelation of daily changes
Repeat above for annual data

In [22]:
```
US10Y['change_rates'] = US10Y.diff()
US10Y['change_rates'] = US10Y['change_rates'].dropna()
US10Y.describe

# Compute and print the autocorrelation of daily changes
autocorrelation_daily = US10Y['change_rates'].autocorr()
print("The autocorrelation of daily interest rate changes is %4.2f" %(autocorrelation_daily))

US10Y.index = pd.to_datetime(US10Y.index)

monthly_rate_data = US10Y['RATE'].resample(rule='M').last()

#annual_data = annual_data.dropna()
# Repeat above for annual data
monthly_rate_data['diff_rates'] = monthly_rate_data.diff()
monthly_rate_data['diff_rates'] = monthly_rate_data['diff_rates'].dropna()

print(monthly_rate_data['diff_rates'])

autocorrelation_monthly = monthly_rate_data['diff_rates'].autocorr()
print("The autocorrelation of monthly interest rate changes is %4.2f" %(autocorrelation_monthly))

US10Y.index = pd.to_datetime(US10Y.index)
annual_rate_data = US10Y['RATE'].resample(rule='A').last()

# Repeat above for annual data
annual_rate_data['diff_rates'] = annual_rate_data.diff()
annual_rate_data['diff_rates'] = annual_rate_data['diff_rates'].dropna()

print(annual_rate_data['diff_rates'])

autocorrelation_annual = annual_rate_data['diff_rates'].autocorr()
print("The autocorrelation of annual interest rate changes is %4.2f" %(autocorrelation_annual))
```

```
The autocorrelation of daily interest rate changes is -0.06
DATE
2014-11-30   -0.17
2014-12-31   -0.01
2015-01-31   -0.49
2015-02-28    0.32
2015-03-31   -0.06
2015-04-30    0.11
2015-05-31    0.07
2015-06-30    0.23
2015-07-31   -0.15
2015-08-31    0.01
2015-09-30   -0.15
2015-10-31    0.10
2015-11-30    0.05
2015-12-31    0.06
2016-01-31   -0.33
```

```
2018-11-30   -0.14
2018-12-31   -0.24
Freq: M, Name: RATE, dtype: float64
The autocorrelation of monthly interest rate changes is -0.02
DATE
2015-12-31    0.10
2016-12-31    0.18
2017-12-31   -0.05
2018-12-31    0.37
Freq: A-DEC, Name: RATE, dtype: float64
The autocorrelation of annual interest rate changes is -0.97
```

Notice how the daily and monthly autocorrelation is small but the annual autocorrelation is large and negative

Are Interest Rates Autocorrelated?

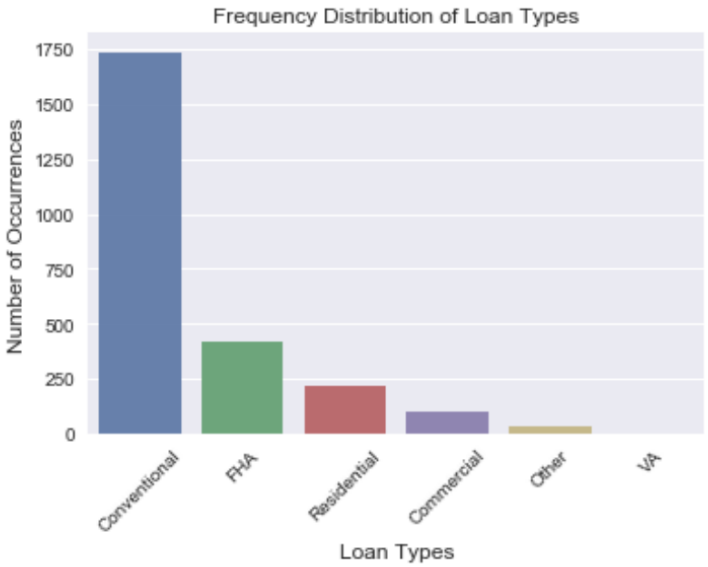
When we look at daily changes in interest rates, the autocorrelation is close to zero. However, if we resample the data and look at annual changes, the autocorrelation is negative. This implies that while short term changes in interest rates may be uncorrelated, long term changes in interest rates are negatively autocorrelated. A daily move up or down in interest rates is unlikely to tell us anything about interest rates tomorrow, but a move in interest rates over a year can tell us something about where interest rates are going over the next year. And this makes some economic sense: over long horizons, when interest rates go up, the economy tends to slow down, which consequently causes interest rates to fall, and vice versa.

One of the really cool things that pandas allows us to do is resample the data. If we want to look at the data by monthly and anually. We can easily resample and sum it up. I'm using 'M' as the period for resampling which means the data should be resampled on a month boundary and 'A' for annual data'. Finally find the The autocorrelation of annual interest rate changes'"

Visual exploration is the most effective way to extract information between variables.

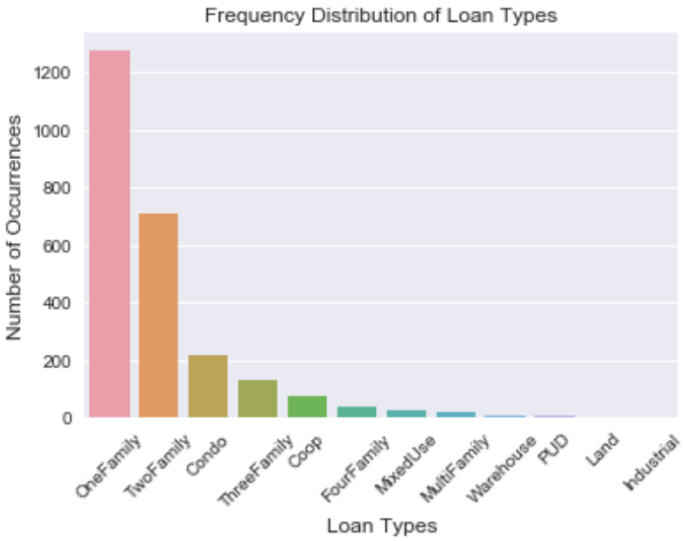
We can plot a barplot of the frequency distribution of a categorical feature using the seaborn package, which shows the frequency distribution of the mortgage dataset column

In [23]:
```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
loan_type_count = data['Loan Type'].value_counts()
sns.set(style="darkgrid")
sns.barplot(loan_type_count.index, loan_type_count.values, alpha=0.9)
plt.title('Frequency Distribution of Loan Types')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Loan Types', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```



Conventional loan type is top market for the Bank then FHA Loan Type

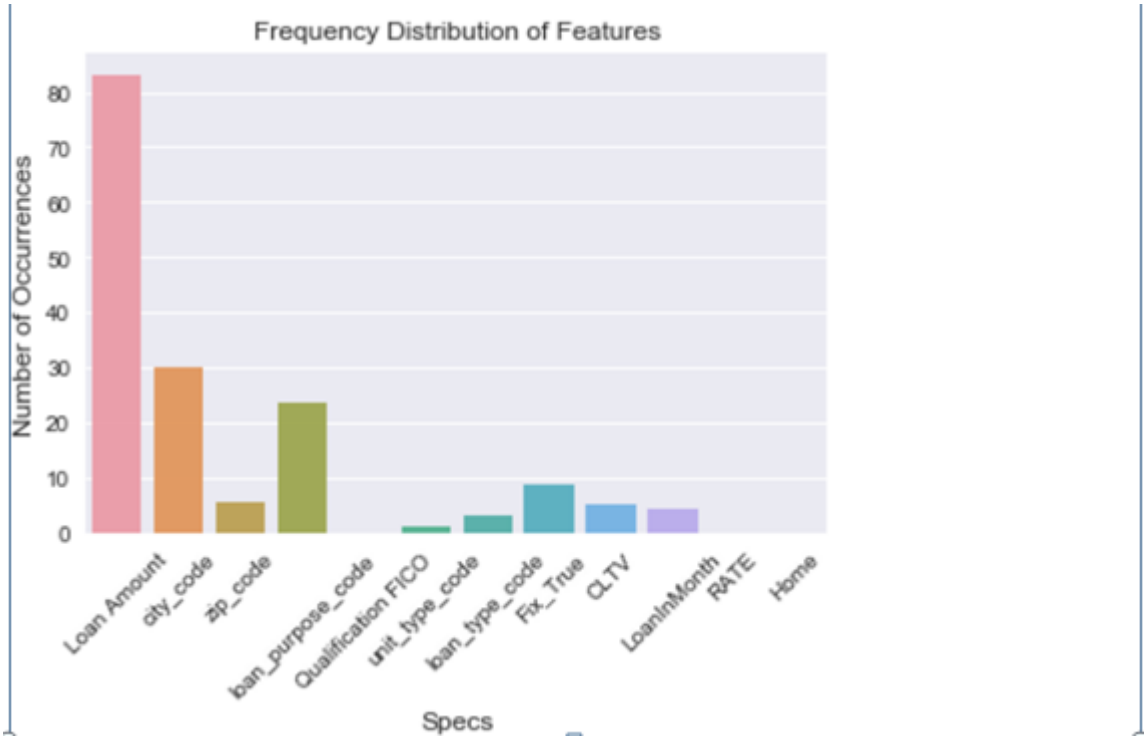
In [24]:
```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
unit_type_count = data['Unit Type'].value_counts()
sns.set(style="darkgrid")
sns.barplot(unit_type_count.index, unit_type_count.values, alpha=0.9)
plt.title('Frequency Distribution of Loan Types')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Loan Types', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```



One Family, Two Family and Condos are the top unit types for the Bank

Univariate Selection
Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features.

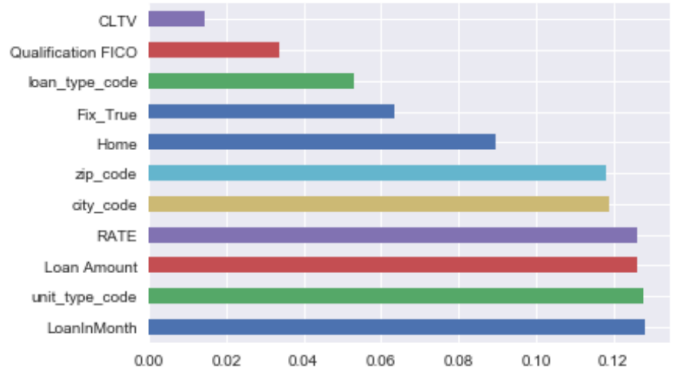
Apply selectkbest class to extract top 10 best features
Concatenate two dataframes for better visualization
Naming the dataframe columns and printing 10 best features



Feature Importance:
The feature importance of each feature of our dataset by using the feature importance property of the model. Feature importance gives us a score for each feature of our data, the higher the score more important or relevant is the feature towards your output variable. Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 10 features for the dataset.

In [78]:
```
model_data['CLTV']=model_data['CLTV'].astype('int64')
import pandas as pd
import numpy as np

X = np.array(model_data.drop(['CLTV'],1))
#X = np.array(model_data.drop(['loan_purpose_code'],1))

y = np.array(model_data['CLTV'])    #target column
Z = model_data.drop(['loan_purpose_code'],1) #Max Col = 10

from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=Z.columns)
feat_importances.nlargest(11).plot(kind='barh')
plt.show()
```

```
[0.12628179 0.11898728 0.11821426 0.03390029 0.1277225  0.05288519
 0.06363727 0.0144152  0.12811489 0.12625031 0.08959101]
```

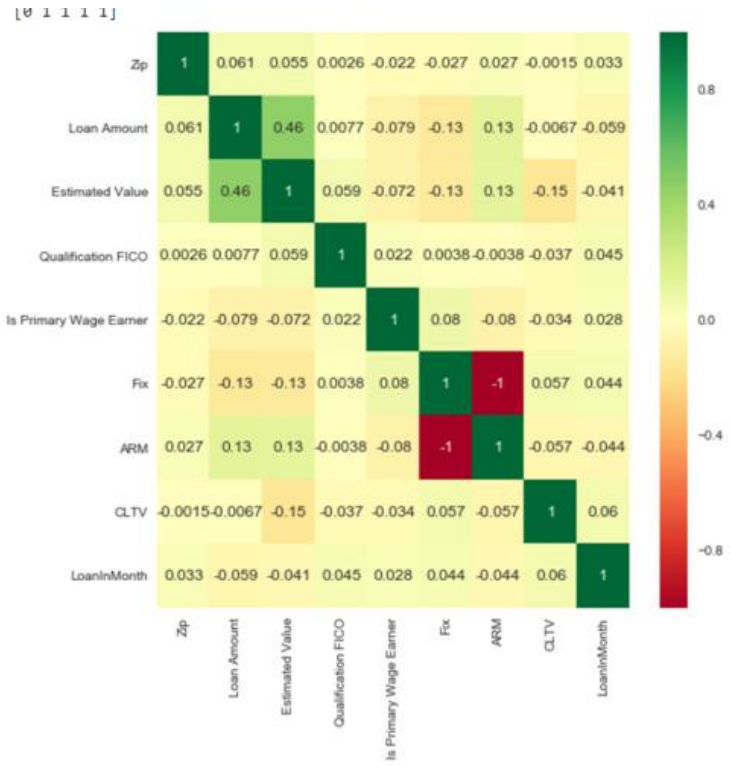


Correlation Matrix with Heatmap:

Correlation states how the features are related to each other or the target variable.

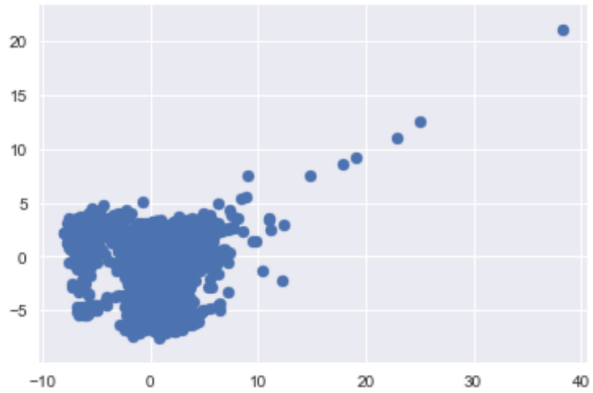
Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable)

Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the seaborn library.

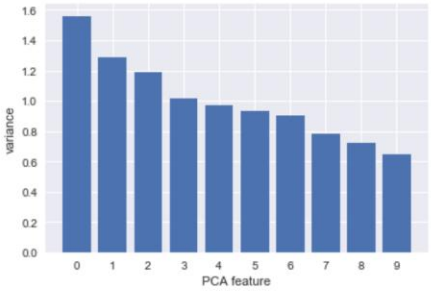


PCA (Principal Component Analysis):
PCA use PCA to de-correlate these measurements, then plot the de-correlated points and measure their Pearson correlation. Compute Pearson correlation coefficient between two arrays.

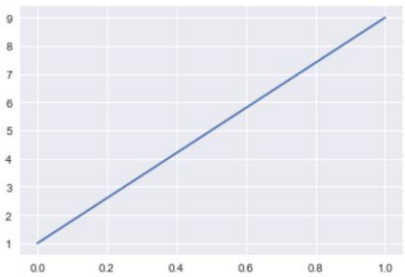
In [80]:
```
def pearson_r(x, y):
    # Compute correlation matrix: corr_mat
    corr_mat = np.corrcoef(x, y)
    # Return entry [0,1]
    return corr_mat[0, 1]

# Import PCA
from sklearn.decomposition import PCA
# Create PCA instance: model
model = PCA()
# Apply the fit_transform method of model to grains: pca_features
pca_features = model.fit_transform(model_data)
# Assign 0th column of pca_features: xs
xs = pca_features[:,0]
# Assign 1st column of pca_features: ys
ys = pca_features[:,1]
plt.scatter(xs, ys) # Scatter plot xs vs ys
plt.axis('equal')
plt.show()
```



In [82]:
```
features = range(pca.n_components_)
feature_names = features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_)

plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(feature_names)
plt.show()

plt.plot([1, 9])
ax = plt.gca()
labels = ax.get_xticklabels()
for label in labels:
    print(label)

pca.fit_transform(X)
print(pca.mean_)
print(pca.components_)
print(pca.explained_variance_)
print(pca.explained_variance_ratio_)
print(pca.singular_values_)
print(pca.n_components_)
print(pca.noise_variance_)
```



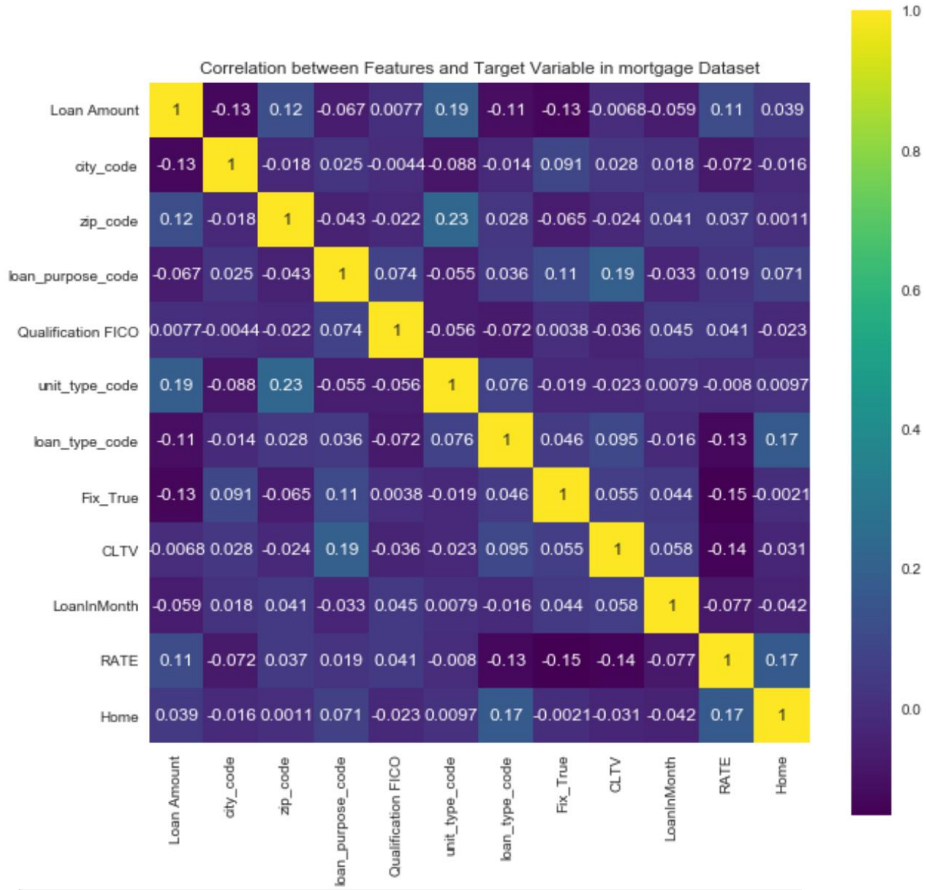

```
Text(0,0,'')
Text(0,0,'')
Text(0,0,'')
Text(0,0,'')
Text(0,0,'')
Text(0,0,'')
Text(0,0,'')
Text(0,0,'')
[4.45099908 2.78884446 2.59613754 7.07676929 1.41143543 6.18192723
 2.66541383 2.27612555 5.84162335]
[[ 6.74631158e-01 -1.42558887e-01  9.47144178e-02  7.15100575e-01
  -6.06926841e-03 -5.83982496e-02 -2.36003998e-02  1.04425183e-02
   9.84355581e-03]
 [ 6.24676971e-01 -3.39271890e-02 -3.63329138e-02 -5.46747569e-01
  -2.79653254e-02  5.54415690e-01 -1.46261205e-02  8.51674717e-04
   1.08518038e-03]
 [ 3.57335110e-01 -6.33451485e-02 -2.18323287e-02 -4.18052138e-01
  -6.39237795e-02 -8.25592495e-01 -7.95718612e-02  2.91204431e-02
   1.36334785e-02]
 [ 1.41995704e-01  9.86633070e-01  2.51691455e-02  5.62644035e-02
  -2.18679814e-02 -4.32461578e-02  1.18712611e-02 -9.30813486e-03
  -4.15073885e-03]
 [ 4.89655127e-02 -2.23000695e-02  5.89480870e-02 -3.00364922e-02
  -3.27211458e-02 -5.76526444e-02  9.93626306e-01 -1.40120500e-02
  -2.90507042e-02]
 [ 3.70608522e-02  1.14319728e-02 -9.90040458e-01  9.93517298e-02
  -6.84259300e-02 -9.86495743e-03  5.66659554e-02 -1.30116095e-02
  -1.68870163e-02]
 [ 3.51813048e-02  1.62475088e-02 -6.29575543e-02 -2.41658456e-02
   7.86270830e-01 -2.69872226e-02  4.38808760e-02  4.33127762e-03
   6.10783295e-01]
 [ 4.11327008e-02  3.19275513e-03 -2.70133316e-02 -1.98134488e-02
   6.03185866e-01 -3.54983945e-02 -8.00519762e-03 -1.40336347e-01
  -7.82511288e-01]
 [-9.97850080e-03  1.28968245e-02 -1.58304568e-02  3.93352125e-03
   8.25158028e-02  1.81686176e-02  1.62052135e-02  9.89382078e-01
  -1.14845823e-01]]
[10.26745402  6.93543827  6.82999328  3.91074715  2.68754743  1.25011306
  0.94337535  0.65752734  0.1609216 ]
[0.30518735 0.20614731 0.20301309 0.11624212 0.07988402 0.03715806
 0.02804066 0.01954419 0.0047832 ]
[160.21434099 131.67610134 130.6712792   98.87804545  81.96870482
  55.9042275   48.5637557   40.54402963  20.05751701]
9
0.0
```



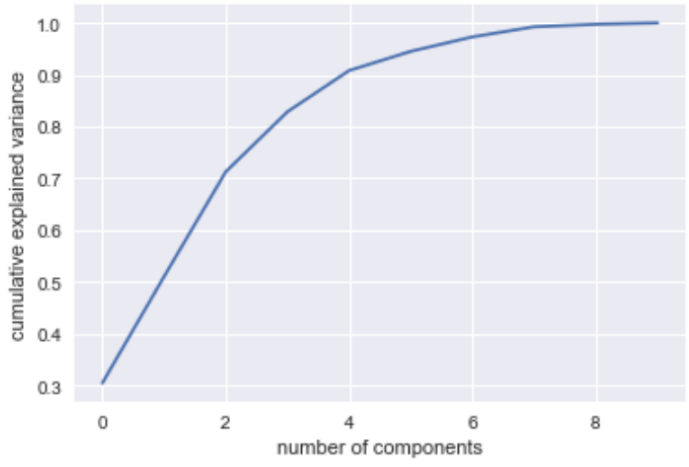
Features with high variance ratio Text(0,0,'Loan Amount') = 4.45
Text(3,0,'unit_type_code') = 7.1 Text(5,0,'Fix_True') = 4.2
Finding Correlation between Features and Target Variable in mortgage Dataset using Heatmap

In [85]:
```
correlation = model_data.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='viridis')
plt.title('Correlation between Features and Target Variable in mortgage Dataset')
```

Out[85]: Text(0.5,1,'Correlation between Features and Target Variable in mortgage Dataset')



In [86]:
```
#Let us load the basic packages needed for the PCA analysis

pca = PCA().fit(fit_data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



We can reduce to six to get the 100% accurary

MODELING

In the Modeling stage we discuss the activities related to the model building part of our project. A selection of five modeling techniques is made that are applicable to our capstone project. From each of these five modeling techniques, a model is built with the feature set provided earlier, and the models are validated using repeated cross-validation.

SELECTION OF MODELING TECHNIQUES

For our modeling we use a combination of predictive techniques. Multiple techniques are selected and applied on the data. For the non-linear regression techniques, we use Support Vector Regression (SVR) and Neural Networks (NN). SVR has shown to obtain excellent performances in regression and time series applications. Neural Networks are a widely used method for time series data that generally gives mixed results.

Another technique we use is Classification and Decision Trees, which is a simple technique that is easy to visualize. Also two ensemble techniques are included, in order to improve the performance of the Classification and Regression Trees. These ensemble techniques are Gradient Boosting Machines (GBM) and Random Forests (RF). These techniques create a multitude of regression trees and select a combination of them in order to maximize the performance.

MODEL BUILDING

Using these five techniques (ARIMA, Linear Regression, Logistic Regression, SVM, SVR, Decision Tree, RF, and KNN) we can create five models. We use the list of features mentioned in section 6.3 as input for our models. A total of 12 features are included, the remaining features were excluded after performing feature selection. For each of the five models hyperparameters were tuned, using grid search. Hyperparameters are the model-specific parameters that are used for optimizing the model. They generally have to be tuned in order to optimize the model's performance, and reduce the variance and bias of the model. By training the model with different values of the size and the decay, and evaluating its performance, we can select the hyperparameters that result in the best performing model in terms of predictive power.

ARMA Model

Estimating an AR Model

We will estimate the AR(1) parameter, ϕ, of one of the Rate, Revenue, Loan_num, series that generated in the earlier . Since the parameters are known for a series, it is a good way to understand the stimation routines before applying it to real data. For monthly_rate_data with a true ϕ of 0.9, we will print out the estimate of ϕ. In addition, we will also print out the entire output that is produced when you fit a time series, so we can get an idea of what other tests and summary statistics are available in statsmodels.

In [89]:
```
US10Y.index = pd.to_datetime(US10Y.index)

monthly_rate_data = US10Y['RATE'].resample(rule='M').last()


# Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA

# Fit an AR(1) model to the first simulated data
mod_rate = ARMA(np.asarray(monthly_rate_data), order=(1,0))
res_rate = mod_rate.fit()

# Print out summary information on the fit
print(res_rate.summary())

# Print out the estimate for the constant and for phi
print("\nWhen the true phi=0.9, the estimate of phi (and the constant) are:")
print(res_rate.params)
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   51
Model:                     ARMA(1, 0)   Log Likelihood                15.789
Method:                       css-mle   S.D. of innovations              0.175
Date:                Mon, 04 Feb 2019   AIC                            -25.579
Time:                        00:15:41   BIC                            -19.783
Sample:                             0   HQIC                           -23.364

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.3612      0.224     10.558      0.000       1.923       2.800
ar.L1.y        0.9057      0.053     16.978      0.000       0.801       1.010
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1041           +0.0000j            1.1041            0.0000
-----------------------------------------------------------------------------

When the true phi=0.9, the estimate of phi (and the constant) are:
[2.3611952  0.90574768]
```

Forecasting with an AR Model

In addition to estimating the parameters of a model, we can also do forecasting using statsmodels. The in-sample is a forecast of the next data point using the data up to that point, and the out-of-sample forecasts any number of data points in the future. These forecasts can be made using either the predict() method if we want the forecasts in the form of a series of data, or using the

plot_predict() method we you want a plot of the forecasted data. We will supply the starting point for forecasting and the ending point, which can be any number of data points after the data set ends. For the simulated series Monthly Interest Rate with ϕ=0.9, we will plot in-sample and out-of-sample forecasts.

Being able to forecast interest rates is of enormous importance, not only for bond investors but also for individuals like new homeowners who must decide between fixed and floating rate mortgages.

There is some mean reversion in interest rates over long horizons. In other words, when interest rates are high, they tend to drop and when they are low, they tend to rise over time. Currently they are below long-term rates, so they are expected to rise, but an AR model attempts to quantify how much they are expected to rise.

In [90]:
```
from statsmodels.tsa.arima_model import ARMA

# Forecast interest rates using an AR(1) model
mod_monthly_rate_data = ARMA(monthly_rate_data, order=(1,0))
res = mod_monthly_rate_data.fit()

# Plot the original series and the forecasted series
res.plot_predict(start=0, end='2020')
plt.legend(fontsize=8)
plt.show()
```

Since we have only used only FOUR years of Monthly Interest Rate Data, we can see the short term downward momentum on the interest rate

A daily move up or down in interest rates is unlikely to tell us anything about interest rates tomorrow, but a move in interest rates over a year can tell us something about where interest rates are going over the next year. The DataFrame daily_data contains daily data of 10-year interest rates from 1962 to 2017

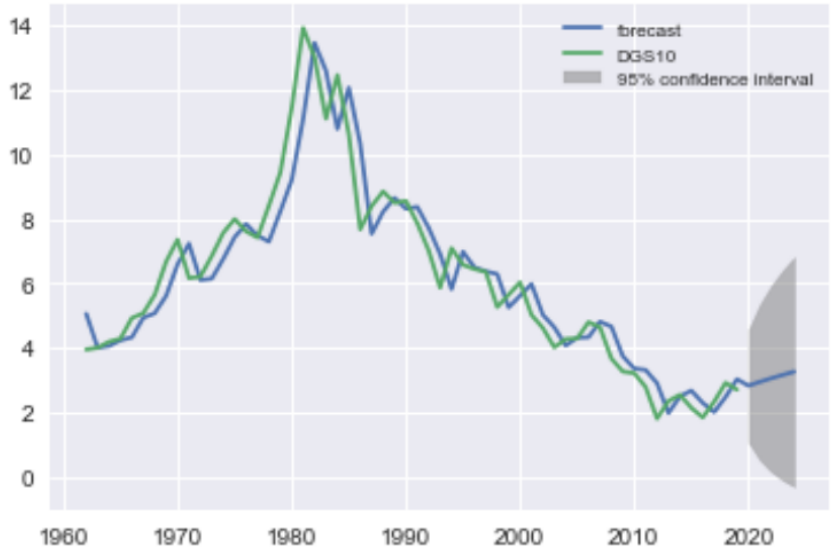
Data cleaning
Convert index to datetime
Repeat above for annual data
Compute and print the autocorrelation of daily changes
Compute and print the autocorrelation of annual changes

```
2018-12-31    0.29
2019-12-31    0.06
Freq: A-DEC, Name: DGS10, dtype: float64
The autocorrelation of daily interest rate changes is 0.07
The autocorrelation of annual interest rate changes is -0.22
```

Notice how the daily autocorrelation is small (0.07) but the annual autocorrelation is large and negative (-0.22)

Import the arma module from statsmodels
Forecast interest rates using an ar(1) model
Plot the original series and the forecasted series

In [92]:
```
annual_rate=daily_data.resample('A').mean()
#type(annual_rate)
annual_rate_data=annual_rate['DGS10']

# Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA

# Forecast interest rates using an AR(1) model
mod_annual_rate = ARMA(annual_rate_data, order=(1,0))
res_annual_rate = mod_annual_rate.fit()

# Plot the original series and the forecasted series
res_annual_rate.plot_predict(start=0, end='2024')
plt.legend(fontsize=8)
plt.show()
```



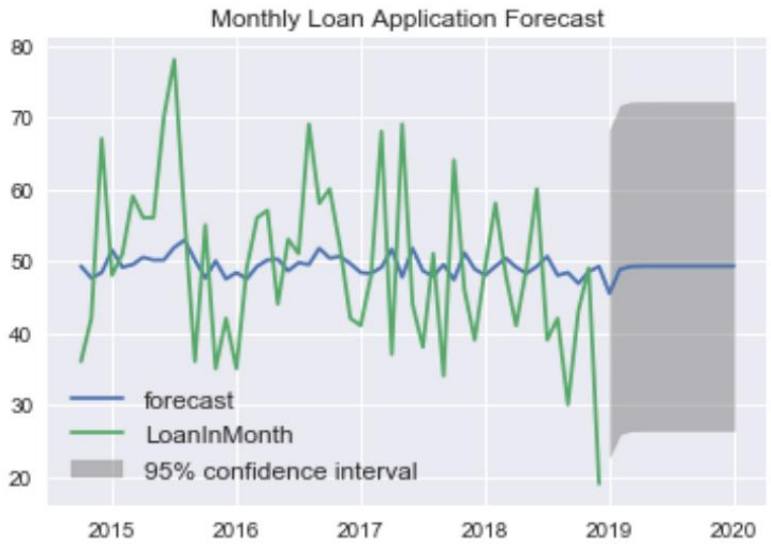
Over long horizons, when interest rates go up, the economy tends to slow down, which consequently causes interest rates to fall, and vice versa.

According to an AR(1) model, 10-year interest rates are forecasted to rise from 2.16%, towards the end of 2017 to 3.35% in five years

Forecast expected monthly closed loans

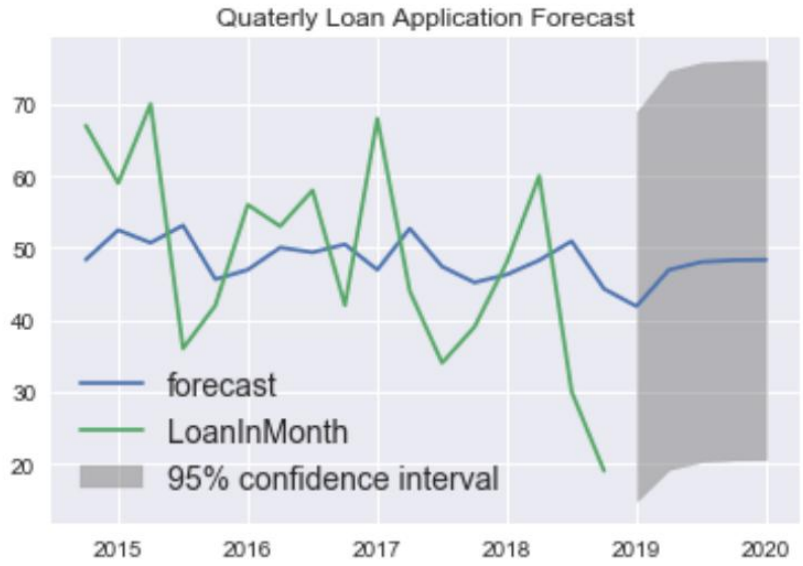
Our Mortgage DataSet contains data from Oct 2014 to December 2018. Let's forecast Monthly Closed Loans and Monthly Revenue. We will forecast monthly_loan_num_data using an AR(1) model

In [93]:
```
## Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA
monthly_loan_num.resample('M').last()
monthly_loan_num_data= monthly_loan_num.resample('M').last()
monthly_loan_num_data=monthly_loan_num_data['LoanInMonth']
# Forecast monthly_loan_num_data using an AR(1) model
mod_monthly_loan_num_data = ARMA(monthly_loan_num_data, order=(1,0))
res_monthly_loan_num_data = mod_monthly_loan_num_data.fit()

# Plot the original series and the forecasted series
res_monthly_loan_num_data.plot_predict(start=0, end='2020')
plt.legend(fontsize=12)
plt.title('Monthly Loan Application Forecast')
plt.show()
```



Above we have plotted the original series and the forecasted series. With 95% confidence interval, Expected Loans per month will be around 50. Low end is 28 & High End is 70

In [94]:
```
## Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA
monthly_loan_num.resample('Q').last()
Q_loan_num_data= monthly_loan_num.resample('Q').last()
Q_loan_num_data=Q_loan_num_data['LoanInMonth']
# Forecast quaterly_loan_num_data using an AR(1) model
mod_Q_loan_num_data = ARMA(Q_loan_num_data, order=(1,0))
res_mod_Q_loan_num_data = mod_Q_loan_num_data.fit()

# Plot the original series and the forecasted series
res_mod_Q_loan_num_data.plot_predict(start=0, end='2020')
plt.legend(fontsize=14)
plt.title('Quaterly Loan Application Forecast')
plt.show()
```



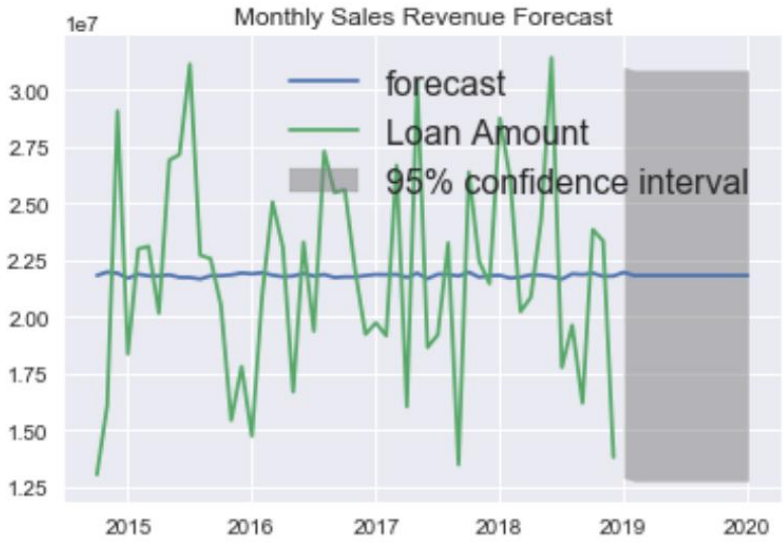
Above we have plotted the original series and the forecasted series. With 95% confidence interval, Expected Loans per month will be around 47. Low end is 20 & High End is 75

Forecast expected monthly revenue

Similarly, we may forecast expected monthly revenue and plot original series and the forecasted series

In [95]:
```
mod_monthly_loan_rev_data = ARMA(monthly_loan_rev_data, order=(1,0))

res_monthly_loan_rev_data = mod_monthly_loan_rev_data.fit()
print("The AIC for an AR(1) is: ", res_monthly_loan_rev_data.aic)
# Plot the original series and the forecasted series
res_monthly_loan_rev_data.plot_predict(start=0, end='2020')
plt.legend(fontsize=16)
plt.title('Monthly Sales Revenue Forecast')
plt.show()
```

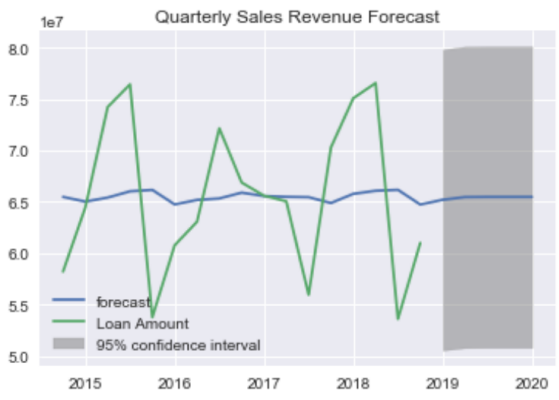
```
The AIC for an AR(1) is:  1715.9386048157016
```



With 95% confidence interval, Expected Revenue per month will be around 22M. Low end is 13M & High End is 32M

Forecasting Quarterly Sales Revenue

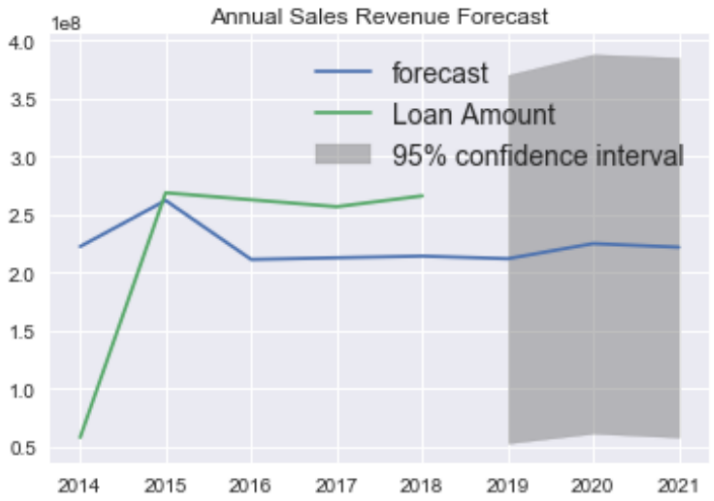
In [96]:
```
loan_patterns['date']= pd.DatetimeIndex(data['Created Date'])

loan_patterns = loan_patterns.set_index('date')
loan_patterns.index

quarterly_revenue_data = loan_patterns['Loan Amount'].resample(rule='Q').sum()

mod_quarterly_loan_rev_data = ARMA(quarterly_revenue_data, order=(1,0))

res_quarterly_loan_rev_data = mod_quarterly_loan_rev_data.fit()

# Plot the original series and the forecasted series
res_quarterly_loan_rev_data.plot_predict(start=0, end='2020')
plt.legend(fontsize=10)
plt.title('Quarterly Sales Revenue Forecast')
plt.show()
```



With 95% confidence interval, Expected Revenue per Quarters will be around 67M. Low end is 52M & High End is 80M'

Forecasting Annual Sales Revenue

In [97]:
```
annual_revenue_data = loan_patterns['Loan Amount'].resample(rule='A').sum()

mod_annual_loan_rev_data = ARMA(annual_revenue_data, order=(1,0))

res_annual_loan_rev_data = mod_annual_loan_rev_data.fit()

# Plot the original series and the forecasted series
res_annual_loan_rev_data.plot_predict(start=0, end='2021')
plt.legend(fontsize=14)
plt.title('Annual Sales Revenue Forecast')
plt.show()
```



In the above graph, we have plotted the original series and the forecasted series. We are expecting little drop in annual mortgage revene. Our current annual revenue is $270M. By end of 2021, with 95% confidence interval, Expected Revenue per Years will be around 220M. Low end is 70M & High End is 360M. With more annual data, we should be able to do better annual revenue prediction.

Linear regression

Purpose of linear regression: Given a dataset containing predictor variables X and outcome/response variable Y, linear regression can be used to:

Build a predictive model to predict future values, using new data X where Y is unknown. Model the strength of the relationship between each independent variable X_i and Y Many times, only a subset of independent variables X_i will have a linear relationship with Y Need to figure out which X_i contributes most information to predict Y It is in many cases, the first pass prediction algorithm for continuous outcomes.

Linear Regression is a method to model the relationship between a set of independent variables X (also knowns as explanatory variables, features, predictors) and a dependent variable Y. This method assumes the relationship

between each predictor X is linearly related to the dependent variable Y.

Independence means that the residuals are not correlated -- the residual from one prediction has no effect on the residual from another prediction. Correlated errors are common in time series analysis and spatial analyses.

```
The train root mean squarred error is : 2.345908703691074
The test root mean squarred error is : 3.963200847042714
The Linear Regression coefficient parameters are : [-0.11919389  0.20419526 -0.08932699  0.17949017  0.16819986 -0.3501712
 -0.68726163  0.04346237 -0.07163668  0.56560897  0.16290692]
The Linear Regression intercept value is : 0.7734251420145459
```

RMSE of the test data is closer to the training RMSE (and lower) if you have a well trained model. It will be higher if we have an overfitted model.

In [99]:
```
from sklearn import metrics # import metrics from sklearn

Rsquared=linear_reg.score(X_train,y_train) # to determine r square Goodness of fit

# how good the model fits the training data can be determined by R squared metric which is here 0.12
Rsquared
print('The R squared metric is :', Rsquared)
```

```
The R squared metric is : 0.12086787994354309
```

Out R^2 = 0.12

The R^2 in scikit learn is the coefficient of determination. It is 1 - residual sum of square / total sum of squares.

Since R^2 = 1 - RSS/TSS, the only case where RSS/TSS > 1 happens when our model is even worse than the worst model assumed (which is the absolute mean model).

here RSS = sum of squares of difference between actual values(yi) and predicted values(yi^) and TSS = sum of squares of difference between actual values (yi) and mean value (Before applying Regression). So you can imagine TSS representing the best(actual) model, and RSS being in between our best model and the worst absolute mean model in which case we'll get RSS/TSS < 1. If our model is even worse than the worst mean model then in that case RSS >TSS(Since difference between actual observation and mean value < difference predicted value and actual observation).

In [100…
```
# K fold cross validation
# cross validation score
cv_score= cross_val_score(LinearRegression(),X,y,scoring='neg_mean_squared_error', cv=10) # k =10
print('cv_score is :', cv_score)

# mean squared error
print('cv_score is :', cv_score.mean())

# Root mean squared error
rmse_cv= np.sqrt(cv_score.mean() * -1)
print('The cross validation root mean squarred error is :', rmse_cv)
```

```
cv_score is : [-10.41761244  -2.87528686  -6.71334008  -6.83023816  -4.73823121
  -3.93130492 -11.82537791  -7.04069096  -3.51224496 -18.03312884]
cv_score is : -7.591745633330166
The cross validation root mean squarred error is : 2.755312256955673
```

With Linear regressor we are able to predict the model with RMSE: The train RMSE = 2.349923482270037 The test RMSE = 3.9655849638809144

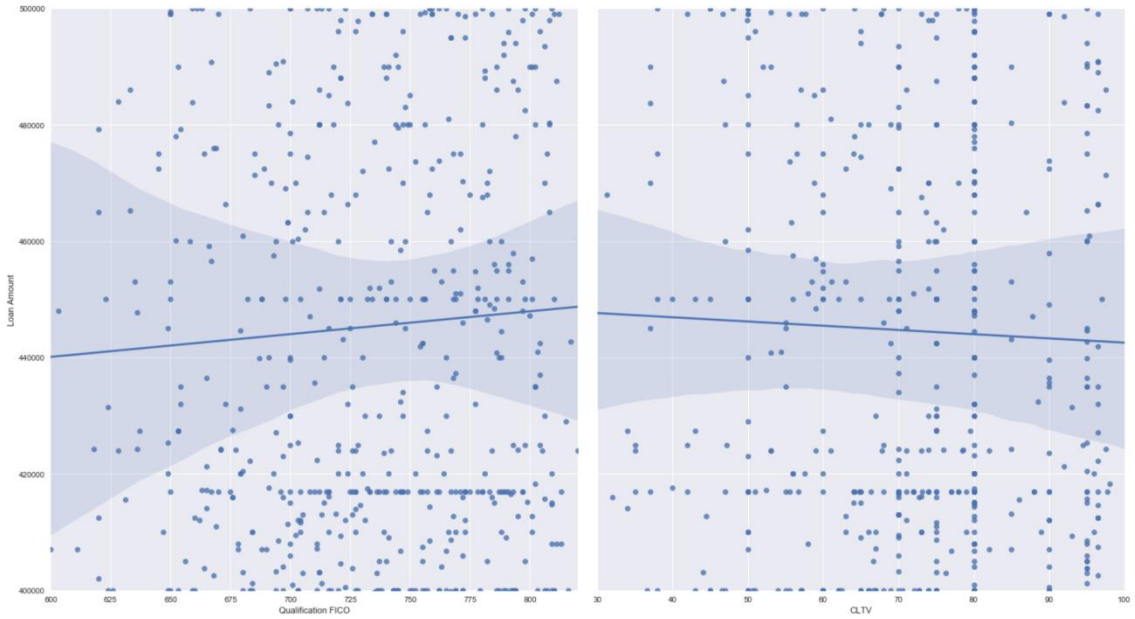
R squared 0.12 and cross validation root mean squarred error is : 2.75

US 10-Years Interest Rate goes up loan amount increases

Home Supply increases, loan amount also increases slightly but eventually housing market will slow down

Interest Rate Chage has bigger impact on Loan Amount compare to Home Supply

```
In [114…
g=sns.pairplot(model_data, x_vars=['Qualification FICO', 'CLTV'], y_vars='Loan Amount', size=11, aspect=0.9, kind='reg')
g.axes[0,1].set_ylim(400000,500000)
g.axes[0,0].set_xlim(600, 820)
g.axes[0,1].set_xlim(30,100)
plt.show()
```

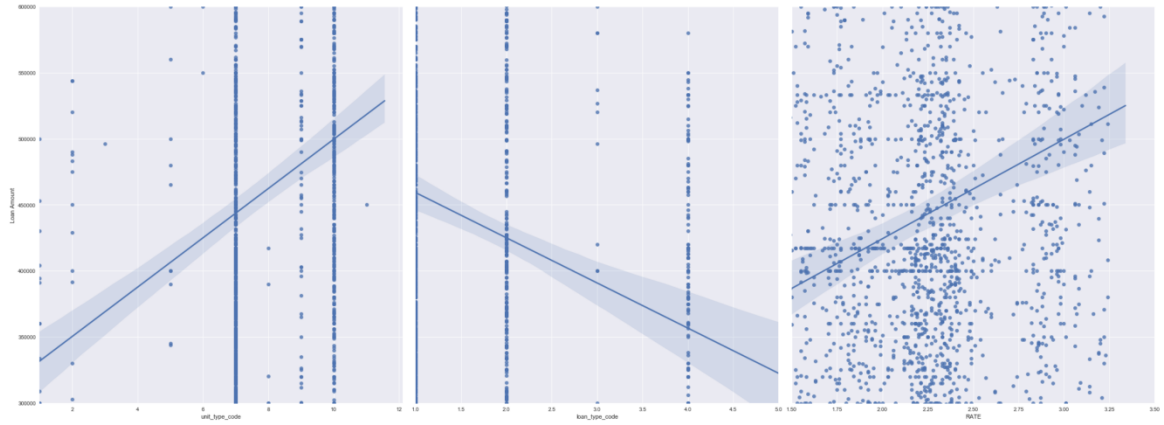


Majority of the FICO scores between 600 and 820

Majority of the CLTV scores between 30% and 100%

FICO goes up, Loan Amount goes up
CLTV goes up, Loan Amount Goes down

```
In [115…
g2=sns.pairplot(model_data, x_vars=['unit_type_code',
       'loan_type_code', 'RATE'], y_vars='Loan Amount', size=11, aspect=0.9, kind='reg')
g2.axes[0,1].set_ylim(300000,600000)
g2.axes[0,0].set_xlim(1.0,12.1)
g2.axes[0,1].set_xlim(1,5)
g2.axes[0,2].set_xlim(1.5,3.5)
plt.show()

print(data_lc['Unit Type'].value_counts(), '\n '  ,data_lc['unit_type_code'].value_counts())
print(data_lc['Loan Type'].value_counts(), '\n ', data_lc['loan_type_code'].value_counts())
```



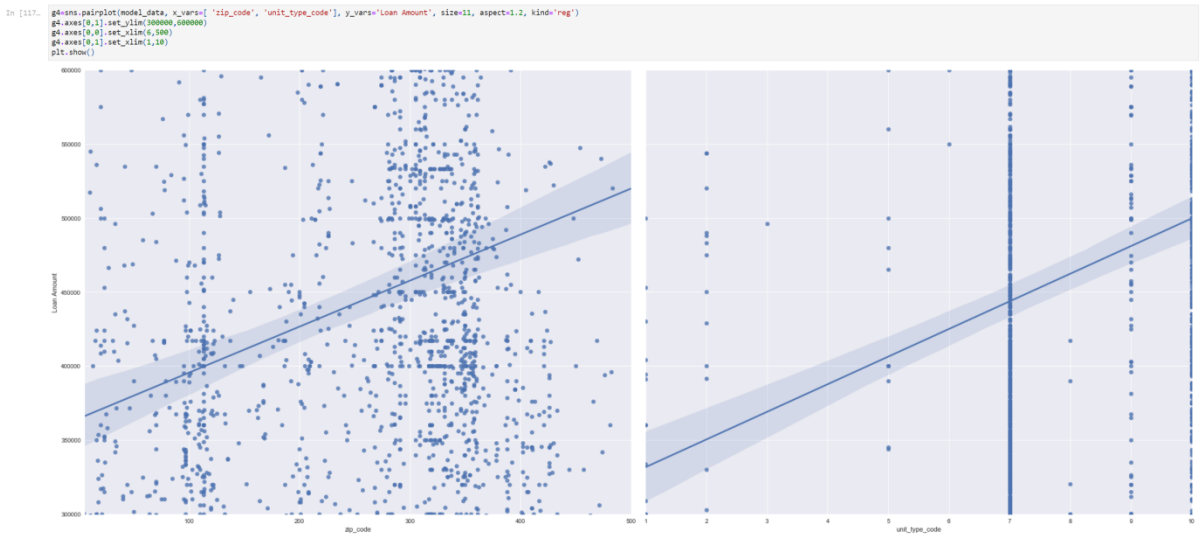
As number of unit decreases ave Loan Amount also decreases

loan_type_code: Conventional is high volume but Slightly low average loan amount

ARM (Adjustable Rate Mortgage) has higher loan amount thnFix_True Rate mortgage Two Family (Code = 10) has higher Loan Amount that three Family (Code = 9)

Conventional Mortgage has Higher Loan Amount FHA ARM has higher Loan Amount compare to Fix Rate Mortgage

NYC NJ (zip_code between 7 & 11) seems to be closing more loans and generation more revenues for the Mortgage Bank. As number of loans per month inceases, loan amount decrease



NYC NJ (zip_code 1st digit starting with 7 & 11) seems to be closing more loans and generation more revenues for the Mortgage Bank. City shows similar results. Population density plays bigger role on Loan Amount. As number of loans per month increases, loan amount decreases

Logistic Regression

In [118…

```
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import train_test_split
from sklearn import preprocessing, cross_validation, neighbors

X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
X_train, X_test, y_train, y_test_knn = cross_validation.train_test_split(X, y, test_size = 0.2)

# Create the DataFrame: numeric_data_only
numeric_data_only = model_data[0:10].fillna(-1000)
# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=21)

# Instantiate the classifier: clf
clf = OneVsRestClassifier(LogisticRegression())

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Print the accuracy
print("Logistic Regression Accuracy: {}".format(clf.score(X_test, y_test)))
```

```
Logistic Regression Accuracy: 0.6866267465069861
```

Logistic Regression Model Accuracy for Loan Types: 69%

Now Centering, Scaling and Logistic Regression and look at the model accuracy

In [119…

```
# Import necessary packages
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from sklearn import datasets
from sklearn import linear_model
import numpy as np

# Load data
X = np.array(fit_data.drop(['loan_type_code'],1))
y = np.array(fit_data['loan_type_code'])
X.shape
y.shape
# Split the data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model and print performance on the test set
lr = linear_model.LogisticRegression()

# fit the model
lr = lr.fit(X_train, y_train)
print('Logistic Regression score for training set: %f' % lr.score(X_train, y_train))
from sklearn.metrics import classification_report
y_true, y_pred = y_test, lr.predict(X_test)
print(classification_report(y_true, y_pred))
```

```
Logistic Regression score for training set: 0.705000
             precision    recall  f1-score   support

          0       0.33      0.10      0.15        20
          1       0.68      0.99      0.80       339
          2       0.50      0.01      0.02        96
          3       0.00      0.00      0.00         5
          4       0.00      0.00      0.00        41

avg / total       0.57      0.67      0.55       501
```

Out of the box, this logistic regression performs little better (70% model accuracy with or without scaling). Lets now scale our data and perform logistic regression:

Random Forests (RF)

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance. Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases.

• Random forests is considered as a highly accurate and robust method because of the number of decision trees participating in the process.

• It does not suffer from the over fitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.

• The algorithm can be used in both classification and regression problems.

• Random forests can also handle missing values

```
Unscalled: Random Forest Classifier Accuracy :  0.7664670658682635
Scalled: Random Forest Classifier Accuracy   :  0.782435129740519
RFC Confussion Matrix:
[[ 10  13   0   0   1]
 [  2 334   9   0   2]
 [  0  42  37   0   0]
 [  0   8   0   1   0]
 [  0  32   0   0  10]]
KNeighborsClassifier Accuracy :  0.782435129740519
RandomForestClassifier :Input Real Data : Conv=5, FHA=4, Res=3, Comm=2:  [[ 3.    1.55  3.17  7.    1.    6.    4.5   1.96  5.4 ]
```

```
RandomForestClassifier :Output of X[975:995]:Conv=5, FHA=4, Res=3, Comm=0: [4 1 2 4 4 4 1 1 1 1 1 4 4 1 1 1 1 1 1 0]
RandomForestClassifier :Output of prediction: Conv=5, FHA=4, Res=3, Comm=: [4 1 2 4 4 4 1 1 1 1 1 4 4 1 1 1 1 1 1 0]
RandomForestClassifier : - Output of prediction: Fix_True =1 & ARM =0: [1 1 1 2 2 1 1 1 2 1 1 2 1 1 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 0 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 0 2 1 4 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 2 1 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 1 1 1 1 2 1 2 1
 1 1 1 1 1 1 1 2 4 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 4 1 1 1 4 1 1 1 3 4 1 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 4 1 1 1 1 1 1 1 2 4 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 2 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 0 1 1 1 1
 1 1 1 1 1 0 2 1 0 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 2 1 2 1 2 2 1 1 1 1 1 2 4 1 4 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 4 1 1 1 1 1]
```

Unscaled: Random Forest Classifier Accuracy : 77%

Scaled: Random Forest Classifier Accuracy : 78%

Both unscalled and scalled for Random Forest Model accuray is almost close

Let us analyze Interest Rate types using Random Forest Classifier.
We have achieved greater accuracy compare to Loan Types. We have Fixed Rate mortgage where Fix_True = 1 &Fix_True = 0 for ARM (Adjustable Rate Mortgage)

```
Random Forest Classifier Accuracy :  0.908183632734531
RFC Confussion Matrix:
[[ 10  40]
 [  6 445]]
RandomForestClassifier Accuracy :  0.908183632734531
```

Random Forest Classifier Accuracy for Fix or ARM is : 0.91 or 91%

Using Random Forest finding Important Features in Scikit-learn

Here, we are finding important features or selecting features in the Mortgage Loan dataset. In scikit-learn, we can perform this task in the following steps:

• First, we need to create a random forests model.

• Second, use the feature importance variable to see feature importance scores.

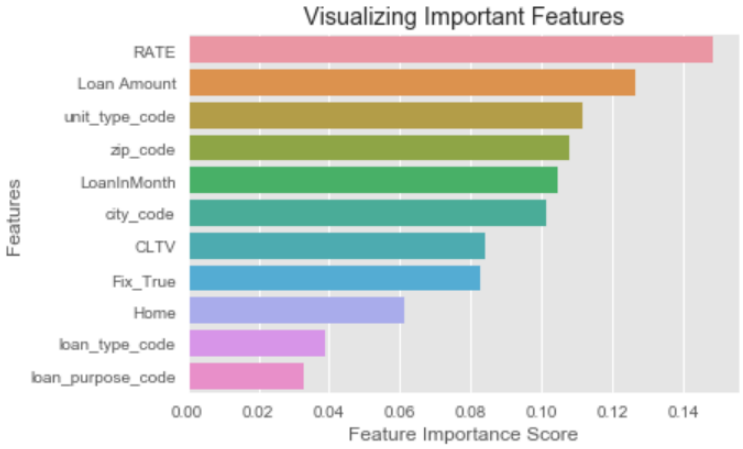
• Third, visualize these scores using the seaborn library.

```
Out[133]: Random Forest Classifier Accuracy :  0.8982035928143712
RATE                0.148499
Loan Amount         0.126198
unit_type_code      0.111566
zip_code            0.107917
LoanInMonth         0.104603
city_code           0.101437
CLTV                0.083989
Fix_True            0.082615
Home                0.061437
loan_type_code      0.038747
loan_purpose_code   0.032993
```

We can also visualize the feature importance. Visualizations are easy to understand and interpretable. For visualization, we have used a combination of matplotlib and seaborn.

```
In [134…
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```



Higher the value, greater the feature importance

Support Vector Regression (SVR)

SUPPORT VECTOR REGRESSION. Those who are in Machine Learning or Data Science are quite familiar with the term SVM or Support Vector Machine. But SVR is a bit different from SVM. As the name suggest the SVR is an regression algorithm, so we can use SVR for working with continuous Values instead of Classification which is SVM.

In [135…

```
X = np.array(model_data.drop(['Loan Amount'],1))
y = np.array(model_data['Loan Amount'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# We want to use svm now
from sklearn import preprocessing, cross_validation, svm
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from matplotlib import style

clf = svm.SVR()
clf.fit(X_train, y_train)
accuracy = clf.score(X_test, y_test)
print('SVM accuracy: ', accuracy)
```

```
SVM accuracy:  -0.025991264219975774
```

SVM-SVR accuracy: -0.026 which is unacceptable for our Mortgage Loan Data Sets

Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which

categorizes new examples. In two dimensional space this hyperplane is a line

dividing a plane in two parts where in each class lay in either side.

```
Support Vector Classifier Accuracy :  0.8962075848303394
SVC Confussion Matrix:
[[  6  47   0   0]
 [  7 392  43   6]
 [  0   0   0   0]
 [  0   0   0   0]]
SVM: cross_val_score accuracy :  [0.9 0.9 0.9 0.9 0.9 0.9 0.9]
```

Support Vector Classifier Accuracy : 0.896 or 90%

k-NearestNeighbors: Predict

Having fit a k-NN classifier, we can use it to predict the label of a new data point. However, there is no unlabeled data available since all of it was used to fit the model! We will use your classifier to predict the label for this new data point, as well as on the training data X that the model has already seen.

```
size of the training feature set is (2000, 11)
size of the test feature set is (501, 11)
size of the training Target set is (2000,)
size of the test Target set is (501,)


Mean of Unscaled Features: 40588.84442139507
Standard Deviation of Unscaled Features: 154733.50862580768


Mean of Scaled Features: -4.132413860696304e-18
Standard Deviation of Scaled Features: 1.0

KNeighborsClassifier Accuracy :  0.8882235528942116
```

KNeighborsClassifierAccuracy : 0.888 or 89%

Decision Tree Algorithm

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

```
Accuracy: 0.8388814913448736
DecisionTre Confussion Matrix:
[[ 20  50]
 [ 71 610]]
Decision Tree Classifier Accuracy :  0.8388814913448736
```

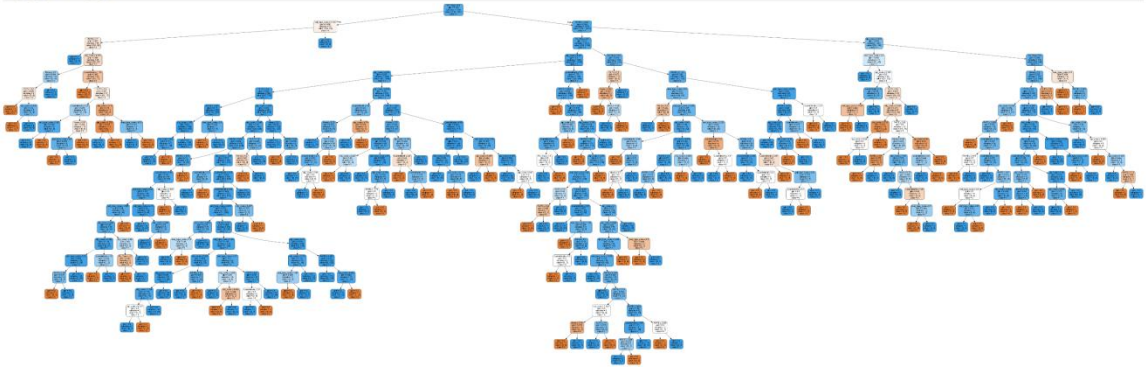
Decision Tree Classifier Accuracy : 0.838 or 84%

Visualizing Decision Trees

We have used Scikit-learn'sexport_graphviz function for display the tree within a Jupyter notebook. For plotting tree, you also need to install graphviz and pydotplus. export_graphviz function converts decision tree classifier into dot file and pydotplus convert this dot file to png or displayable form on Jupyter Notebook.

```
col_names = (fit_data.columns)

X = (fit_data.drop(['Loan Amount'],1))
#X = (model_data.drop(['Qualification FICO'],1))
#X = (model_data.drop(['Home'],1))
y = (fit_data['Loan Amount'])

feature_cols = X.columns

fit_data.columns

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('c:/scripts/mvb/mortgage_dt.png')
Image(graph.create_png())
```

Out[147]:



In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning.

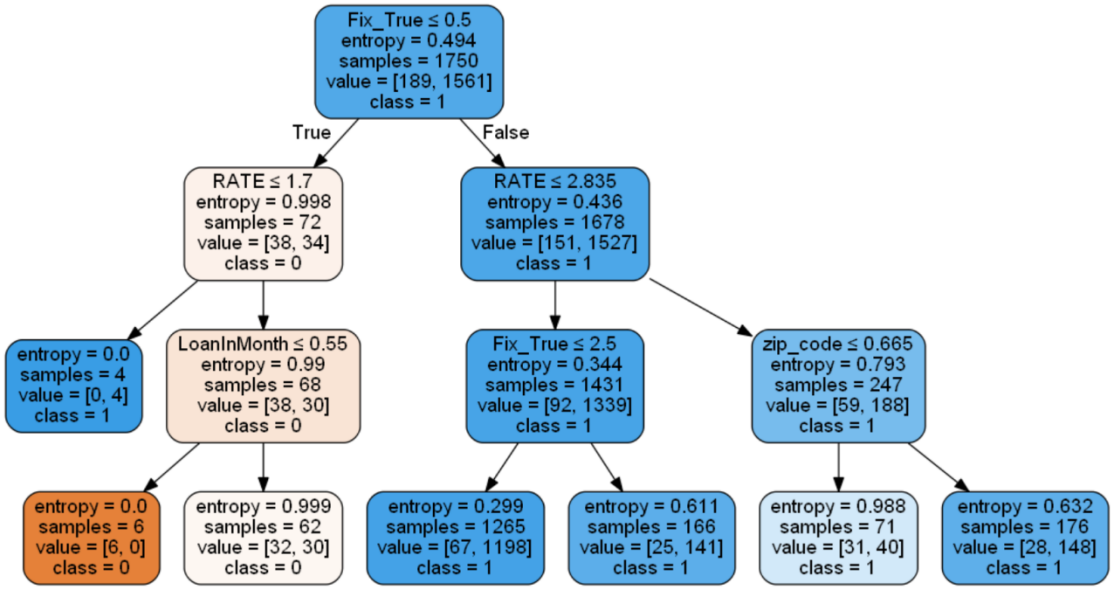
We can plot a decision tree on the same data with max_depth=3. Other than pre-pruning parameters, We can also try other attribute selection measure such as entropy.

In [153…

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

#Visualizing Decision Trees
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1','2'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('c:/scripts/mwb/mortgage_dt.png')
Image(graph.create_png())
```

```
Accuracy: 0.9041278295605859
```

Out[153]:

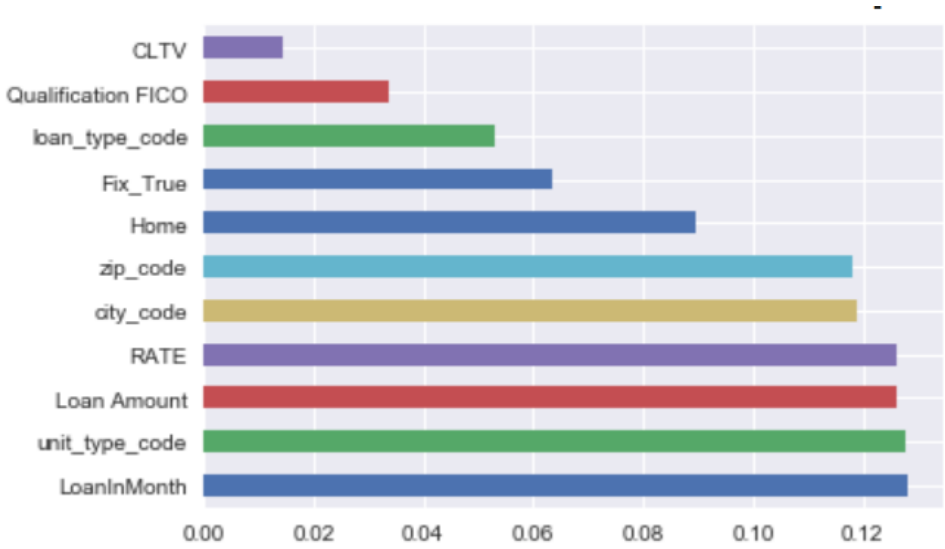


Plotted a decision tree on the same data with max_depth=3. Easy to visualize..

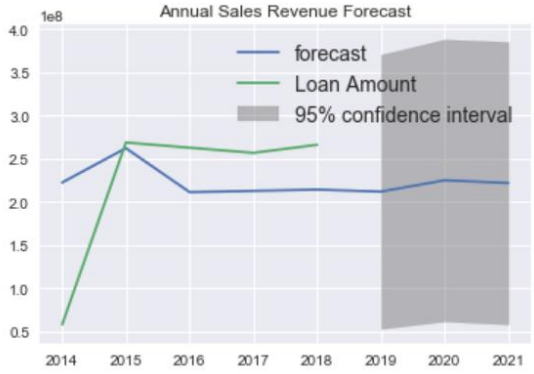
Decision tree model accuracy has also improved.

6. Results and Conclusions

Among the features in the dataset, we have identified the most important features.



After forecasting the annual sales revenue using ARIMA model,



In the above graph, we have plotted the original series and the forecasted series. We are expecting little drop in annual mortgage revene. Our current annual revenue is $270M. By end of 2021, with 95% confidence interval, Expected Revenue per Years will be around 220M. Low end is 70M & High End is 360M. With more annual data, we should be able to do better annual revenue prediction.

The results of various classification models are as follows:

Linear regression:

- R^2: 0.12, As number of unit decreases ave Loan Amount also decreases
- loan_type_code: Conventional is high volume but Slightly low average loan amount
- ARM (Adjustable Rate Mortgage) has higher loan amount than Fix_True Rate mortgage Two Family has higher Loan Amount that three Family Conventional Mortgage has Higher Loan Amount FHA ARM has higher Loan Amount compare to Fix Rate Mortgage

Logistic regression: Accuracy – 68.9%

Random Forest: Accuracy – 91%

KNN – Accuracy – 89%

Decision Tree – Accuracy – 84%

SVM-SVR accuracy: -0.026

SVM – Accuracy – 90%

From the models performed our Random forest has provided the highest accuracy and the worst performed models are SVM and linear regression.

7. REFERENCES

- .(https://www.sciencedirect.com/science/article/pii/S0957417419308176)
- https://blend.com/blog/challenges-solutions/mortgage-automation-platform/
- https://www.automationanywhere.com/solutions/financial-services/mortgage-process-automation
- https://www.docsumo.com/blog/mortgage-automation
- https://www.kaggle.com/datasets/gauravduttakiit/mortgage-bank-loan